Policy-based Approach for Context-aware Systems

Ph.D Thesis

Mohammed H. Al-Sammarraie

This thesis is submitted in partial fulfillment of the requirements

for the degree of Doctor of Philosophy

Software Technology Research Laboratory

De Montfort University

Leicester - United Kingdom

September 2011

Dedication

To my father

Husam Al-Sammarraie

Who was the motivation behind this Ph.D thesis and every goodness in me

Thank you for all the sacrifices you have done and still doing

To my mother

Manal Al-Izzi

For all the prayers, unconditional love and faith in me

Thank you for everything you have done for me since I was born

To my wife

Mariam Bani Saad

For her endless love, support and believing in me

Thank you for being there even during the hardest of times

Abstract

Pervasive (ubiquitous) computing is a new paradigm where the computers are submerged into the background of the everyday life. One important aspect of pervasive systems is context-awareness. Context-aware systems are those that can adapt their behaviours according to the current context. Context-aware applications are being integrated into our everyday activity aspects such as: health care, smart homes and transportations. There exist a wide range of context-aware applications such as: mobile phones, learning systems, smart vehicles.

Some context-aware systems are *critical* since the consequence of failing to identify a given context may be catastrophic. For example, an auto-pilot system is a critical context-aware system; it senses the humidity, clouds, wind speed and accordingly adjusts the altitude, throttle and other parameters. Being a critical context-aware system has to be provably correct.

Policy-based approaches has been used in many applications but not in context-aware systems. In this research, we want to discover the anatomy (i.e. architecture, structure and operational behaviour) of policy-based management as applied to context-aware systems, and how policies are managed within such a dynamic system.

We propose a novel computational model and its formalisation is presented using the Calculus of Context-aware Ambients (CCA). CCA has been proposed as a suitable mathematical notation to model mobile and context-aware systems. We decided to use CCA due to three reasons: (i) in CCA, mobility and context-awareness are primitive constructs and are treated as *first-class citizens*; (ii) properties of a system can be formally analysed; (iii) CCA specifications are *executable*, and thus, leading to rapid prototyping and early validation of the system properties. We, then show how policies can be expressed in CCA. For illustration, the specification of the event-condition-action (ECA) conceptual policy model is modelled in CCA in a natural fashion.

We also propose a policy-based architecture for context-aware systems, showing its different components, and how they interact. Furthermore, we give the specification of the policy enforcement mechanism used in our proposed architecture in CCA.

To evaluate our approach, a real-world case study of an *infostation-based mobile learning (mLearning) system* is chosen. This mLearning system is deployed across a university campus to enable mobile users to access mobile services (mServices) represented by course materials (lectures, tests and tutorials) and communication services (intelligent message notification and VoIP). Users can access the mServices through their mobile devices (Hand-set phones, PDAs and laptops) regardless of their device type or location within a university campus. We have specified the mLearning system in CCA (i.e. specification based on policies of the mServices), afterwards, the specification is simulated using the CCA interpreter tool.

We have developed an animation tool specially designed for the mLearning system. The animation tool provides graphical representation of the CCA processes. In

terms of *safety* and *liveness*, some important properties of the mLearning system have been validated as a proof of concept.

Declaration

I declare that the work described in this thesis is original work undertaken by me for the degree of Doctor of Philosophy, at the software Technology Research Laboratory (STRL), at De Montfort University, United Kingdom.
No part of the material described in this thesis has been submitted for any award of any other degree or qualification in this or any other university or college of advanced education.
This thesis is written by me and produced using LaTeX.

Mohammed Al-Sammarraie

Publications

1. Mohammed Al-Sammarraie, François Siewe and Hussein Zedan. Formalising Policies of a mLearning System Using CCA. *In Proceeding of the 4th ACM International Workshop on Context-Awareness for Self-Managing Systems*, CASEMANS'10 - Copenhagen, Denmark, pages 2:11–2:19, 2010.

2. Mohammed Al-Sammarraie, François Siewe and Hussein Zedan. Formal Specification of an Intelligent Message Notification Service in an Infostation-based mLearning System using CCA. *To appear in the 2011 International Conference on Current Trends in Information Technology*, CTIT'11 - UAE, 2011.

Acknowledgments

First and foremost, my truthful thankfulness goes to the most merciful **ALLAH** for all the things he blessed me with throughout my whole life, without those blessings, I would not be here standing in this position at all.

After studying for three degrees (including this one), at three universities, in three different countries, I have learned one important thing - I could never have done any of this, particularly the research and writing that went into this thesis, without the love, support and encouragement of a lot of people.

Most importantly, I would like to thank my mentor, supervisor and the one behind this project, **Professor Hussein Zedan**, whom without his support, encouragement and guidance this thesis would not have been possible to achieve. I am so happy that I was able to finish my Ph.D under his supervision. The love and care he offered to his students, including me, has affected this work in so many good ways.

Also, many thanks and gratitude goes to my supervisor **Dr. François Siewe**. For his critical comments, technical suggestions and professional guidance have always improved this thesis since day one.

I want to express my deepest thanks to the one who all the good words would

not be enough to describe him, whom without, I would not be the man I am today, to my beloved father **Husam**. What he has done for me from big sacrifices to simple advices is beyond the limits. He was the one to push me forward, to guide me, and to plant every goodness in me. I owe it all to you father. I hope one day I can be the man just the way you are.

Also, I would like to thank my mother **Manal** for her endless support, prayers and guidance since the day I was born. Thank you for being the perfect mother a person can wish for, thank you Mom for everything.

Moreover, my innermost thanks and appreciation goes to my lovely, beautiful and amazing wife **Mariam**. Her love, care, advice, prayers and encouragement have always enhanced my heart, body and soul to the furthest. Thank you for sharing the happy and sad times, the good and bad times with me. Thank you for being you.

I also like to acknowledge my son *Husam Jr.* for when looking into his eyes and playing with him gave me the happiness I needed through the bitter times. I also would like to thank my only sister *Hiba* for she was the greatest sister anyone can have. In addition, I would like to thank my parents-in-law *Prof. Mahdi Hameed Abood* and *Dr. Eman Mohammed Ali* for their support and prayers. I also would like to acknowledge my brother-in-law *Dr.Omar* and sisters-in-law *Sarah* and *Marwa*.

I would like to thank my dear friend *Dr. Ali Al-Bayatti*, for he was the one who helped me first when I started pursuing my PhD, for his support throughout its years. I also want to thank my very dear friend and office mate *Dr. Murad Magableh*, for the many discussions and laughs we had. Not forgetting to thank my friends *Dr. Nasser Al-Alwan* and *Dr. Mohammed Taye*.

I would like to thank my dearest friends and family whom life has scattered among different continents, thanks for everyone who showed his/her support throughout the years.

Last but not least, I would like to thank every member of the STRL for providing the academic and home-like environment and the support whenever needed, especially *Mrs. Lynn Ryan* and *Mrs. Lindsey Trent*.

Contents

List of Figures

List of Tables

List of Abbreviation

AAA	Authentication, Authorisation and Account
AC	Access Control
AMS	Agent Management System
BLP	Bell and LaPadula
BRSs	Bigraphical Reactive Systems
CAP	Context-Aware Policy
CAS	Context-Aware System
CASS	Context-Aware Sub-Structure
CCA	Calculus of Context-aware Ambients
ccaPL	Calculus of Context-aware Ambients Programming Language
CCS	Calculus of Communicating Systems
CE	Context Expression
CoBrA	Context Broker Architecture
CRBAC	Context Role Based Access Control
DAC	Discretionary Access Control
DUMMBO	Dynamic Ubiquitous Mobile Meeting Board

ECA	Event Condition Action
ECG	ElectroCardioGram
ER-Model	Entity Relationship Model
GPS	Global Positioning System
GSM	Global System for Mobile Communications
HBR	Human Behavioral Recognition
HRU	Harrison, Ruzzo and Ulman
IETF	Internet Engineering Task Force
IMN	Intelligent Messaging Service
IP	Internet Protocol
IS	Infostation
ISC	Infostation Centre
JADE	Java Agent Development Framework
JavaCC	Java Compiler Compiler
JCAF	Java Context-Awareness Framework
JDK	Java Development Kit
LAN	Local Area Network
MA	Mobile Ambients
MAC	Mandatory Access Control
MANETs	Mobile Ad hoc Networks
MMS	Multimedia Messaging Service
MO	Managed Objects

mService	Mobile Service
NPD	Named Protection Domain
OBJ	a programming language
OS	Operating System
PA	Personal Assistant
PBM	Policy Based Management
PBNM	Policy Based Network Management
PC	Personal Computer
PDA	Personal Digital Assistant
PDP	Policy Decision Point
PEP	Policy Enforcement Point
PMT	Policy Management Tool
PPG	PhotoPlethysmoGraph
PR	Policy Repository
QoS	Quality of Service
RBAC	Role Based Access Control
SIP	Session Initial Protocol
SKT	Skin Temperature
SMS	Short Message Service
SN	System Network
UML	Unified Modelling Language
VDM	Vienna Development Method

VoIP	Voice Over IP
WiMAX	Worldwide Interoperability for Microwave Access
WLAN	Wireless LAN
WWW	World Wide Web
Z	The name for a formal method

Chapter 1

Introduction

Objectives:

- Give an introduction and the motivation of this research
- List the research questions
- Present the research methodology
- Give the thesis structure

1.1 Overview

"*The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it*" [103]. This statement is quoted from Mark Weiser's visionary article 'The Computer for the Twenty-First Century'. This article was published in 1991 and it was considered as a head of its time for it has shed the lights on the essential aspects of context-awareness, mobility and availability in pervasive computing (aka ubiquitous computing).

The revolution of computing technology has underwent two waves; firstly was the mainframes which each was shared by lots of people, succeeded by the personal computers, in which each person has his own computers. However, Ubiquitous computing defines the next (third) wave of technology in which computers are submerged into the background of the everyday life. Pervasive computing is considered as the new paradigm for the next generation of distributed computing, where the users and devices can be mobile and the computations are context-aware. The term refers to the increasing integration of information and communication technology into people's lives and environments. Pervasive computing has many potential applications, from health and home care to environmental monitoring and intelligent transport systems. One important aspect of pervasive computing is *context-aware systems*.

Context-aware systems are those which can sense different aspects of their environment and use the contextual information to adapt their behaviour. Dey *et al.* have best defined context as: 'any information that can be used to characterise the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user

and applications themselves'. For that, context-aware systems have to be able to do three key processes; (i) sense the context in its environment, (ii) reason about changes in context and (iii) react to those changes.

Some context-aware systems are *critical* since the consequence of failing in identifying and reacting to a given context may be catastrophic. For example, an autopilot system is a critical context-aware system; it senses the humidity, clouds, wind speed and accordingly adjusts the altitude, throttle and other relevant parameters. Being a critical context-aware system has to be provably correct.

Policy-based management is an approach that is used to simplify the management of a system by establishing policies to deal with situations that are likely to occur. Policies of a system are operating rules that can be referred to as a way to maintain a system's consistency. For example, suppose a bank system that have a policy against authorising the bank employees to access customers' accounts. Each time that situation arises, the system can refer to the policy, rather than having to make decisions on a case-by-case basis.

1.2 Motivation

Many researchers have been working in the area of context-awareness for the last twenty years, and many frameworks and architectures have been proposed to suit different aspects of context-aware systems. However, none of the existing approaches to context-awareness has used policy-based management approach in context-aware systems. Policies are essential component of a context-aware system, as it is for every system, for policies define the way a system behaves when dealing with different situations. As context is continuously changing; a context-aware system is

of a dynamic nature, thus affecting the behaviour of the system. Subsequently, the system may change its policies according to the changes in context.

The work in this thesis introduces the policy-based management approach to context-aware systems, by proposing a novel *policy-based architecture for context-aware systems*. Our approach comes from the need to probable change in the policies of a context-aware system without changing the actual architecture of the system to suit the new policies.

1.3 Thesis Scope and Research Question

The work in this thesis focuses on the following issues:

- **Context-aware systems:** firstly, this research focuses on context-aware systems in particular. It sheds the light on what is a context-aware system, how it works and what are its requirement. However, this research also investigates the advances and some of the challenges regarding the area of context-awareness. As context-awareness is a newly emerged field of research, there still are some lack in regards to architecture and linguistic support.

- **Policy-based management:** secondly, this research sheds the light on the area of policies and policy-based systems. It gives some definitions of a policy and presents a policy terminology with focusing on policy types. In addition, this research explores policy-based management.

- **Formalism in context-aware system:** finally, this research investigates the area of formalisation in context-aware systems. For testing is a partial and informal technique that can not cover all aspects of the system; correctness of a system can only be ensured by using a sound formal method to prove

wanted properties of the system. This research considers the lack of formal support towards context-awareness and then investigates the current available formal methods and decides which of which is suitable to model context-aware applications.

In the light of the above arguments regarding the scope of this thesis, the main research question can be formulated as follows:

How can policy-based management system be used in the development of context-aware systems?

The major goal of the work in this thesis is to address the above question efficiently. However, dividing the research question into sub-questions would makes it an easier job to address each one separately. The sub-questions can be summarised as follows:

- How can policies be specified in the Calculus of Context-aware Ambients (`CCA`)?
- Can a context-aware system be designed based on its policies?
- How can policy-based management be integrated into a context-aware system?

1.4 Research Methodology

The methodology used in the work of this research is scientific research method, in which it describes constructive research where the term 'constructive' refers to the contribution in knowledge being developed as a new architecture, model, technique, etc. However, it is difficult to carry out a scientific research in a specific field efficiently without a good knowledge of the structure of the material available in the same field. Therefore, part of carrying out such research must be devoted to

acquiring this knowledge [105].

The methodology of this research was done using four work packages. The first work package concerns the research literature review, the second one addresses the computational model on which the work in this thesis is based. The third work package elaborates the proposed architecture. The last work package concerns with the evaluation of the work.

- **Work package 1:** Research background

  The research background mainly consists of a literature review which elaborates the aspects that are related to the research question. However, many resources have been used such as: digital libraries, books, articles, etc.

- **Work package 2:** Computational model

  At this stage, the research investigates the computational model on which the rest of the work in this thesis will be based. Then it investigates the available formal methods to specify context-aware systems, and elaborates the suitable mathematical notation. At last, the chosen formal method is used to specify different aspects related to the scope of this thesis.

- **Work package 3:** Architecture

  This work package concerns the design of the architecture to capture the objectives of this research as elaborated in the research question. During this stage, the research investigates previous context-aware systems architectures / frameworks and their limitations, and the policy-based management approach in order to specify the objectives of the proposed architecture and its contribution.

- **Work package 4:** Evaluation

  In the final stage, a real-world case study of a context-aware system is chosen,

then modelled and formalised using the discussed formal method. At last, the case study model is mapped onto the proposed architecture and wanted properties of the system is proven used the execution environment of the formal method.

1.5 Measure of Success

The success criteria for the reported work in this thesis is as follows:

- The research questions set at the beginning of this research have to be met,
- A study showing how our proposed architecture is different from others has to be done,
- A study of why `CCA` was chosen among other formal methods has to be done,
- A study that shows the advantages of using policy-based management has to be achieved.

1.6 Thesis Structure

According to the objectives of the work, this section provides a glance of the remaining chapters of this thesis along with a summary of their contents:

Chapter 2: Literature Review

At first, this chapter presents an overview of context-aware systems. It starts with what is context, why using context, what and how to capture context, what is context-awareness, how to represent context and what is a context-aware system. This chapter then examines the context-aware system development process and investigates the challenges that lie within each phase. Afterwards, it gives some ex-

amples of existing context-aware systems which most are location-aware. Last but not least, this chapter presents a general overview on policies. It looks at the definitions of a policy, when to use policies and what are the types of policies in terms of their terminology, and what are their uses. Finally, it presents what is a security policy, access control policy and most importantly what is a management policy. We then presented the ECA model which is used to represent policies in general and management policies in specific. The information in this chapter is presented cumulatively for the sake of better understanding.

Chapter 3: Computational Model

This chapter is divided into three parts. The first part presents the computational model on which we model the work in the thesis; the computational model is based on the notion of ambient, which can model any entity in a system. The second part represents the formal part which presents `CCA`. `CCA` has been proposed recently as suitable mathematical notation used to model mobile applications that are context-aware. The third and last part of this chapter presents the specification of policies using a suitable formal method. It investigates how policies, regarding their type, can be specified using `CCA`.

Chapter 4: Policy-based Architecture for Context-aware Systems

This chapter proposes a novel policy-based approach to develop context-aware systems. This chapter first discusses the conceptual five-layered framework of context-aware systems which our approach is based on. Then it investigates the policy-based management standard which we integrated into our approach. Finally, it presents a specification of the policy enforcer used in our architecture.

Chapter 5: Formalisation

Using CCA, this chapter provides the specification of the policies of a mLearning system (Full case-study details are given in Appendix B). It first proposes a model for the mLearning system, then it formalises that model based on the previously specified policies of the mobile services offered by the system.

Chapter 6: Tool Support

This chapter is divided into three parts. The first one presents the CCA interpreter, which is a CCA code execution tool based on the reduction semantics of CCA. It also presents the syntax of ccaPL which is a machine readable version of the syntax of CCA. The second part of this chapter proposes an animation tool based on the CCA interpreter. The last part presents the validation of some of the mLearning properties, in which it executes some user's behaviour along with the mLearning system's specified processes to check whether it executes the right processes at the right time or not.

Chapter 7: Conclusion and Future Work

This chapter presents a summary of the work in this thesis, and summarises the results. Then it draws some conclusions and gives suggestions for future work that falls from this work in the light of the limitations of our approach.

Chapter 2

Literature Review

Objectives:

- Present a literature on context, context-awareness and context-aware systems
- Investigate some existing context-aware systems
- Explore what is a policy and its terminology

2.1 Overview of Context-aware Systems

2.1.1 Context

It is a challenging task to define the word 'context' as many researchers tried to give their own definition of what context is. The word itself, derived from the Latin *con* (with or together) and *texere* (to weave), depending on this, [15] defined context as an active process of how humans give their environment a meaning by weaving their experiences together with it.

[91] Stated the first definition for context and it described it as location, identities of people and objects around. While [85] referred to context as the user's identity and location, environment and time. Using synonyms is also a way of defining context, such as (Background, situation, or circumstance), but that is a wide range of definitions. [19] gave a definition which is: context is the computer's knowledge of the elements of the user's environment.

Context, in The Free On-Line dictionary, is that which surrounds, and gives meaning to something else [92]. According to [93], context is the abstract level of the current situation derived from the physical and logical sensors. Schilit et al. [90] claim that the important aspects of context are: where you are, who you are with, and what are the available resources.

In the same manner, context is the physical and conceptual states of interests to a particular person, object or place [79]. Context is, simply, the environmental information according to [86]. With all the definitions given, it is very hard to elucidate the definition of the word 'context', but, Dey [34] made it easier to understand

the word 'context', among all the definitions above, by describing it as:

> *Context is any information that can be used to characterise the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.*

2.1.2 Why Using Context

Context helps users to identify other users, locations, actions and events that determine everyday social behaviour. Context is considered so beneficial when applied in some systems, such as the ones that consider change in location [81]. For example, mobile tour guides has been designed to get the user familiarized with a new location (s)he is in [5]. Active badge project [102] has been used to track where the employees are (inside a company) and that can help in forwarding the phone-calls for them. Context if captured and processed the right way can provide helpful and useful information and services to the users.

2.1.3 Capturing Context

2.1.3.1 What to capture

A single context atom can be described with two attributes:

- Context type: such as time, location, ID.
- Context value: means the raw data gathered by the sensor

Some attributes might be added to be captured as the above two are not sufficient in some cases of context-aware system [8]:

- Time stamp: contains date and time of when the attribute was senses.

- Source: contains the ID of the source (i.e. physical sensor).
- Confidence: describes the accuracy of the captured data depending on the source. Some sources may deliver in accurate information like some of the location tracking tools.

The process of capturing context has its limitations, there are two main problems when it comes to data collection:

- Cost: the cost of employing sensors and other devices in order to generate the context data to be capture, nevertheless securing this data is somehow massive.
- Privacy: most of the people are sensitive about capturing their information (who, when, where...etc) and does not have the willingness to publicise their information wherever they are.

2.1.3.2 How to capture

For a system, context-awareness means that the system has to be aware of the environment it is running in, and can use its information to adapt itself to provide information or service to the user.

As mentioned before, context is the environmental information available nearby a system. Capturing information (contextual sensing) is the basic level of context-awareness [79]. Context is captured via sensors. Sensor is not a word referencing to the hardware sensors only; however, it refers to every data source that may provide a useful data to the system. Sensors can be classified into three different categories [8]:

- *Physical Sensors*: *aka* hardware sensors.

- *Virtual Sensors*: is another way of acquiring data via software applications. It is possible to determine the location of a student in a university not only through hardware sensor (GPS for example), but also through the processes of logging in/out onto the university's computers.

- *Logical Sensors*: this type of sensors combines context information from both physical and virtual sensors to solve bigger problems. For example, combining the GPS information of a person and his login/logout processes information in a building can tell us the exact location of that person.

Physical sensors are the most commonly used type of sensor, below, is a list of some common physical sensors [93]:

- *Light Sensor*: single optical sensor, that supplies useful information on the light density, reflection, wave length and type of light (natural, artificial).

- *Camera*: supply a wide range of visual information about the environment, shape and object recognition. Cameras are commonly used due to their ease of use and low cost.

- *Microphones*: supply very interesting information about the type of input (noise, speaking or music), the noise level and with a higher power consumption, microphones can provide speech recognition.

- *Location*: provides information about the location of the user outdoor through GPS or GSM. While indoor, a system like the Active Badge [12] can provide this type of information.

- *Temperature*: it gives very useful information when used in some application, especially in detecting body heat.

- *Touch sensors*: is a type of a switch that only operates if touched by an object.

- *Motion Detector*: a device that contains a physical mechanism or electronic sensor that operates in the presence of a moving object within its range.

Many types of sensors can be integrated and deployed within a system to give it the sense of context-awareness. According to [10], a study has been made to show what are the most used sensors in pervasive computing and what are the percentage of usage of each type, it is summarised in (Figure 2.1) below:

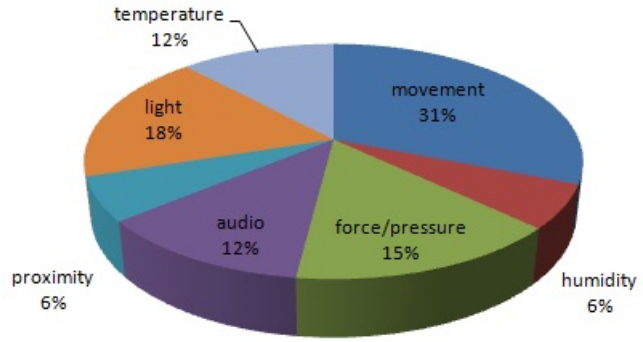


Figure 2.1: Sensors types used in pervasive computing

2.1.4 Context Model

The context is a raw data that needs to be processed and stored, to achieve this goal, a context model is needed to define and store context in a machine processable form. Strang *et al.* [98] has summarised the relevant context modelling approaches as follows:

1. **Key-value model:** is the simplest form of context modelling, such as: name-Sam, time-03:00p.m., location-home. This model is frequently used for it is easy to manage, however, it lacks capabilities for refined structuring for sufficient context retrieval algorithms.

2. **Markup scheme model:** is a hierarchical data structure which consists of markup tags with attributes and contents. The contents of the markup tags are recursively attached to other markup tags.

3. **Graphical model:** the Unified Modelling Language (UML) is a general modelling tool which has an efficient graphical tool. UML's generic structure makes it suitable to model context. This kind of modelling is applicable to derive an Entity Relationship ER-model from it, which makes it useful to structure tools for relational database in a context management system architecture.

4. **Object oriented model:** the dynamism of context is a problem in a context-aware system, thus, using this approach aims at encapsulation and reusability. In this model, the object level encapsulates the details of context processing, and access to the contextual information is done via specified interfaces only.

5. **Logic based model:** in this model, the context is defined as facts, expressions and rules, for which concluding expressions is derived from a set of other expressions. Common to all logic based models is a high degree of formality.

6. **Ontology based model:** ontologies, in general, are able to specify concepts and interrelations, and thus, context. A good example on modelling context using ontologies is the Context Broker Architecture CoBrA system [24], which provided a set of ontological concepts to characterise entities (e.g. persons, places, devices).

2.1.5 Context-aware Systems

Context-aware systems are those which have the ability to adapt their operations to the current context, thus aim at more usability and effectiveness by considering environmental context [8, 87]. Schilit and Theimer [91] first defined Context-Aware Computing in 1994 as the ability of a mobile application to interact with the information existing in its environment. Also, Context-Aware System was defined as the ability of a system to sense, detect, interpret and respond to the nearby context [85, 79, 20], and use the environmental information to provide information and/or

service to the users, regarding their needs [34].

Context-aware applications provide information and/or services to the users. For a context-aware application, the process of providing information and services can be of three different classes according to [57]:

- Presenting information and services [P]: either an information is presented to the user or a selection of several actions to perform services
- Automatic service execution [E]: here, the system takes the appropriate action and perform a service on behalf of the user.
- Attaching context information for later retrieval [T]: refers to application that tags captured data with relevant context information.

In context-aware systems, the captured context fall under four basic categories: *Identity, Location, Activity [106] and time [36].* These four categories are considered as a primary context that answers only the questions of who, what, where and when. But they also act as indexes to other information (i.e. secondary context) [36]. For example: from a user's identity, many other related information can be acquired such as phone number, address and list of friends. Also, by obtaining a user's environment we can determine what other objects and persons are near the user and what activities are occurring in the nearby. These information represent what is meant by secondary context.

A context-aware system has to deal with context information in the following manner: (i) *sense* the context, (ii) *reason* about context information and (iii) *act* upon the context information.

2.1.6 Cataloguing Previous Works on Context-awareness

Dey and Abowd [36] made a fair piece of work describing previous systems and applications that consider context-awareness (Table 2.1). They categorised the context type of each system into: Activity, Identity, Location and Time (in which they will be referenced as A, I, L and T respectively). Also they categorised the context-awareness into three features: *Presentation* (P), *Execution* (E) and *Tagging* (T).

Table 2.1: Application of context and context-aware categories [36]

SYSTEM NAME	**SYSTEM DESCRIPTION**	Context Type				Context-aware		
		A	**I**	**L**	**T**	**P**	**E**	**T**
Classroom 2000 [6]	Capture of a classroom lecture			X	X			X
Cyberguide [6]	Tour guide		X	X		X		
Teleport [18]	Teleporting	X	X	X			X	
Stick-e Documents [19, 20, 21]	Tour guide		X	X	X	X		X
	Paging and reminder	X	X			X		X
Reactive Room [26]	Intelligent control of audiovisuals	X	X	X			X	
GUIDE [31]	Tour guide			X		X		
CyberDesk [38, 35, 33]	Auto integration of user services	X				X	X	
Conference Assistant [39]	Conference capture and tour guide	X	X	X	X	X		X
Responsive Office [41]	Office environment control			X	X		X	
NETMAN [43, 62]	Network maintenance			X		X		
Fieldwork [85, 79, 80]	Fieldwork data collection			X	X	X		X
Augment-able Reality [83]	Virtual post-it notes			X		X		X
Context Toolkit [86]	In/Out Board		X	X	X	X		
	Capture of serendipitous meetings		X	X	X		X	X
Active Badge [102]	Call forwarding		X	X		X	X	

2.2 Context-Aware System's Development

Context-aware systems have the same basic phases of any system's development life cycle which are: requirement, specification, design, implementation and testing. However, the validation phase is an important phase for the development of a context-aware system [108], as the nature of such a system must guarantee the sensors reliability. Figure 2.2 below depicts a context-aware system development

steps.

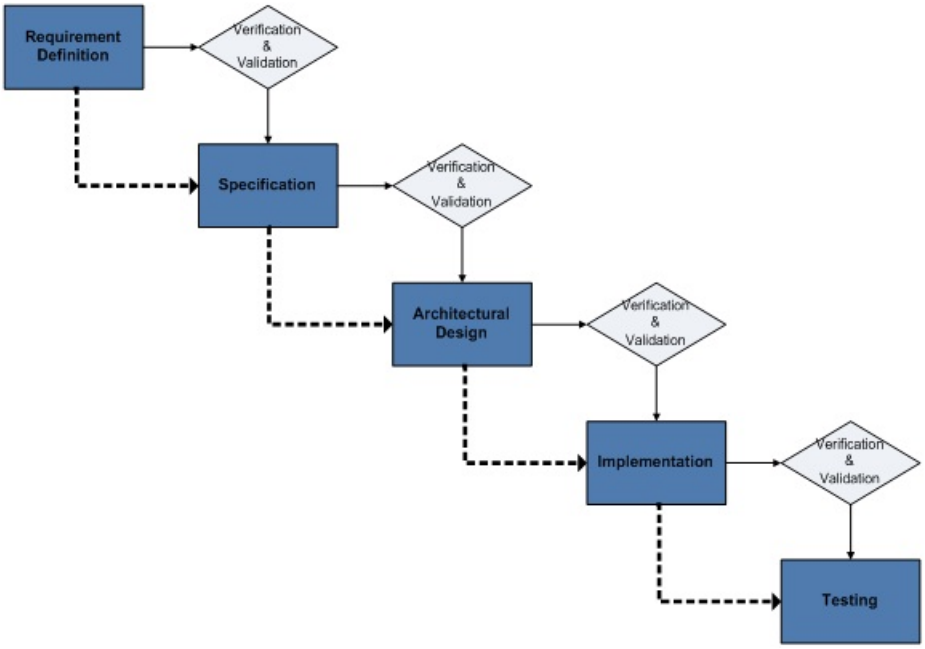


Figure 2.2: Context-aware system development

2.2.1 Requirements Definition

It is the task of determining the needs of the users or defining the system outputs to meet the users' requirements. [76] stated that there are eleven heuristic general methods used to determine what any context-aware system should offer to its users. The eleven heuristics are given below:

- Separation of concerns: sensors, services and applications should be separated from one another. A change in one should not affect the others.
- Flexibility and openness: the system should be flexible and open to add new components, change (upgrade) existing components or even integrate other components that are not related to the system.
- Scalability: the system's complexity should depend on the number and complexity of rules not on the number of sensors, actuators or users.

- Support of reuse: the system's approach should provide the reuse of contextual information previously collected (context database). Also, components developed for one system should be suitable for reuse in other systems.

- Support during development, debugging and deployment: the system should be supportive for the development and deployment cycle as well as the debugging and error tracing.

- Explanations and accountability: the system should offer the users the information that makes them able to understand system actions. Also clear information should be provided on how the system behaves and a clear explanation why the system behaves that way.

- Security and privacy: the system should be provided with security and privacy. Such as encryption and authentication. These two parameters should be provided at the architecture level.

- Reliability: the operation of the system should be reliable and reflect the intention of the users and the designers.

- Match between the context-aware system and the real world: the system should be designed and built in a way that guarantees an interaction between the context-aware system and the real world. The system should change its information or status according to changes detected in the real world.

- Manual override: the users should have the privilege for manual overriding decisions based on contextual information taken by the system.

- Reconfiguration and management: the system should be easy to configure, reconfigure and manage when deployed. It should be possible to easily change and adapt the system behaviour during runtime.

2.2.2 Specification

The objective of this phase is to develop methods to support the earlier phase of a context-aware system development. This phase allows writing a notation using a modelling language (formal method) to specify the user's requirements.

Formal specifications use mathematical notation to specify the wanted properties a system must have, precisely, without unduly constraining the way to achieve these properties. They describe what the system must do but not how it is to be done. Such an abstraction makes formal specifications useful in the process of developing a system, because they confidently answer the questions of what the system does [97].

Formal methods are used for several reasons which are [107]:

- Ambiguities, omissions and contradictions can be found in the informal formulation of the problem throughout the formalisation process.
- The formal model can be proven correct with mathematical methods.
- Analysis can be made to a formally specified system to have or not to have wanted properties.
- A formal specified system can be embedded within a larger system with more confidence.
- The formal model (partly) leads to automated development methods and tools like simulations.
- Formally specified systems' designs can be easily compared with each other.
- It might be a good idea to draw the structure of a program or a system to make the understanding of the interaction processes inside it easier, as well as

the interaction between its components [1].

However, there are five general classes for formal methods which are:

- *Model-based*, such as Z notation [40] and Vienna Development Method (VDM) [64]
- *Algebraic-based*, such as OBJ [50]
- *Process calculus*, such as π-Calculus [69, 101] and Calculus of Communicating Systems (CCS) [68]
- *Logic-based*, such as Temporal and Interval logic [16]
- *Net-based* such as Petri-nets [101, 17]

The specification of a context-aware system requires special constructs due to the nature of such a system, such as context-awareness and mobility. There are some limitations in some formal methods which make them not suitable to model context-aware systems. For example, the Z notation and the CCS modelling languages do not support modelling of context-aware objects [11]. Another example is the π-Calculus model, which is beginning of providing languages and analytical tools for business processes with mobile structure.

Another example is the Bigraphs modelling language. Bigraphs is similar to the UML; but it is more specialised for the specification of distributed and mobile systems [72, 74, 51]. A Bigraphs model consists of two graphs (hence the name), a place graph and a link graph - sharing nodes. The place graph is considered to be like a tree. While the link graph can be a *hypergraph* [74, 1], which means a link graph can connect two or more objects. The idea behind Bigraphs is to model the locational structure of the system with the place graph. The topography of a system

can usually be modelled as objects. The connections between objects are modelled using links. For example: two connected object via a link are able to communicate with each other. For example: (Figure 2.3) below shows an example of Bigraphs and how the objects communicate between one another and with the outside (through x, y). Bigraphical Reactive Systems (BRSs) [71] is an extension of the Bigraphs that aims at modelling concurrent and mobile systems. Birkedal *et al.* [13] attempted the modelling of context-aware systems using BRSs. However, they concluded that BRSs are not suitable for directly modelling context-aware applications.

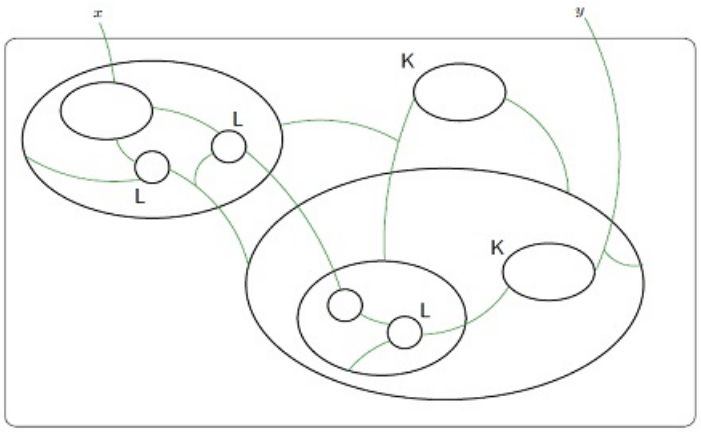


Figure 2.3: An example of Bigraph [55]

However, for such lack of formalism support, some other modelling formal method have to be used to meet the requirements of a context-aware system. The Calculus of Context-aware Ambients [1] (`CCA`) is a formal method (process calculus) suitable for describing mobile systems that are context-aware. `CCA` considers context-awareness and mobility as primitive constructs, which both are important features of a context-aware system.

`CCA` is built upon the Mobile Ambients (MA) from which it inherits the mobility capabilities. However, `CCA` provides new constraints to enable mobile ambients and

[1] `CCA` will be discussed further in chapter 3

processes to have a sense of awareness of the environment they are being executed in. The logical formula that is used to express context in CCA is called context expression [95]. An example on how to model using CCA is that, consider a smart mobile phone which diverts all incoming calls when the user is with a friend. The CCA model is as follows:

$$phone\ [!\ \texttt{user\_with}(friend)?\ \texttt{switch\_to} < divert > .\mathbf{0}]$$

2.2.3 Architectural Design

It is the process or phase of defining the technical solution (framework, architecture, components, modules, interfaces, input and output data for the system, etc) to satisfy the requirements of a system.

A general framework for a context-aware application has been suggested by [37, 7]. Figure 4.1 depicts the five-layered conceptual framework. The lowest level, the physical layer, consists of sensors (Physical, Virtual and Logical). The second level, the data layer, it is responsible for the retrieval of raw context data. The third level, the semantic layer, contains objects which transform the captured data into meaningful and useful information. This layer raises the result of the data layer to a high abstraction level, however, it is not implemented in every context-aware system [8, 65]. The fourth level, the inference layer, it is responsible for storing and managing the information from the semantic layer. The higher level, the application layer, uses the logical information from the previous layer to perform a specific action.

[60] has proposed a context managing framework, which is useful to overcome the problem of small memory and small processors in mobile devices. Figure 2.5 depicts the context managing framework. Four main functional entities comprise

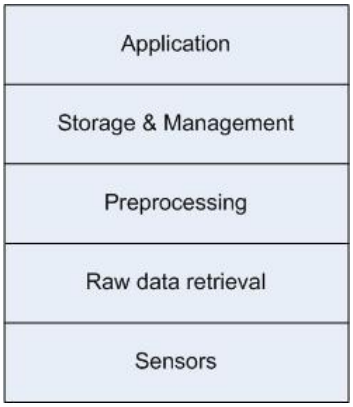


Figure 2.4: The five-layered context-aware system's framework [8, 7]

this context architecture: the context manager, the resource servers, the context recognition services and the application [61].

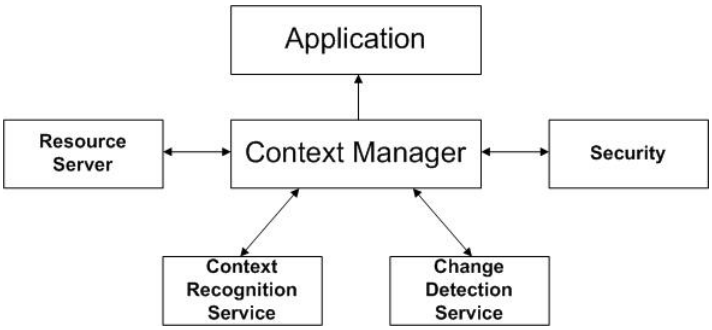


Figure 2.5: Context managing framework [60]

Yet, another architecture was proposed by [42] as part of a project called Context-Awareness Sub-Structure (CASS), the architecture is depicted in Figure 2.6. The middleware contains an Interpreter, Context Retriever, Rule Engine and Sensor Listener [8, 42]. The gathered information from the sensors located in sensor node are stored in the database by the Sensor Listener, Context Retriever is used to retrieve the stored context data. The interpreter services can be accessed by both the Context Retriever and the sensor listener. Location Finder, Change Listener and sensors have the built-in communication capabilities.

Another architecture based on the five-layered framework was built as part of the hydrogen project [53]. The project was especially made for mobile devices. The

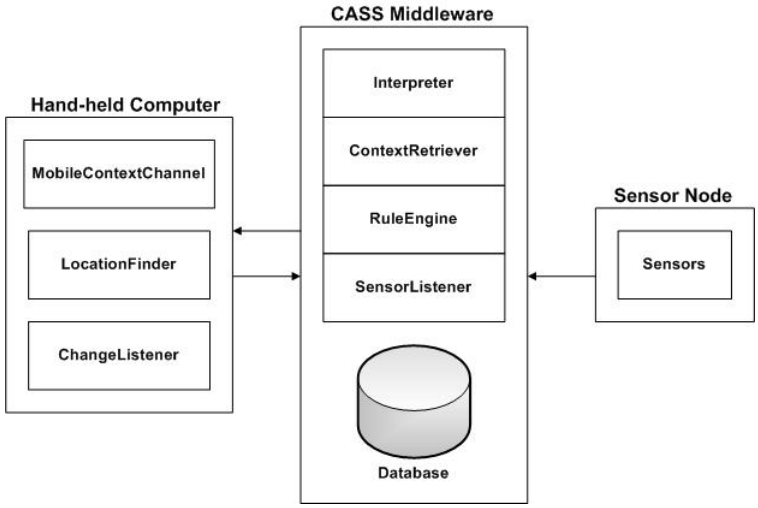


Figure 2.6: The CASS Architecture [42]

hydrogen approach is to make the system able to distinguish between a local and a remote context, when the local context is the information a device is aware of; while the remote context is the information another devices are aware of. The exchange of these contexts is done via WLAN, Bluetooth, etc. This is called *Context Sharing*.

The architecture of the hydrogen project consists of three layers (Figure 2.7) which are all located on the same device [8, 53]. The lowest layer, the adapter layer, is responsible for retrieving raw context data captured by the sensors. The second layer, the management layer, used to provide contexts to the client application and to retrieve contexts. It is also called Context Server. The top layer which is the Application layer, used to implement the appliance code on.

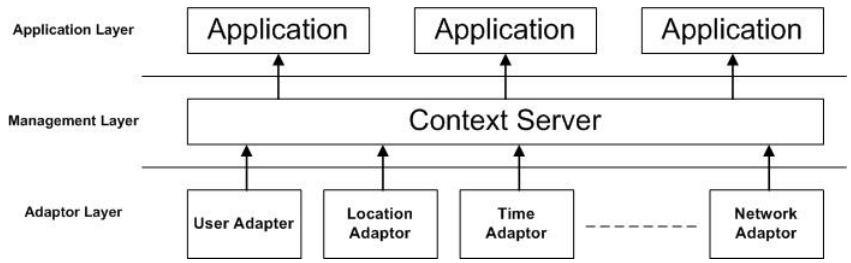


Figure 2.7: Hydrogen's project architecture [53]

A sentient object [53] is an encapsulated entity which mainly consists of three components: sensory capture, context hierarchy and inference engine, as shown in Figure 2.8. The sentient object communicates sensors and actuators, where sensors produce software events, while actuators consume software events. A sentient object can be both producer and consumer to another sentient object. However, a graphical tool is available to help building a sentient object.

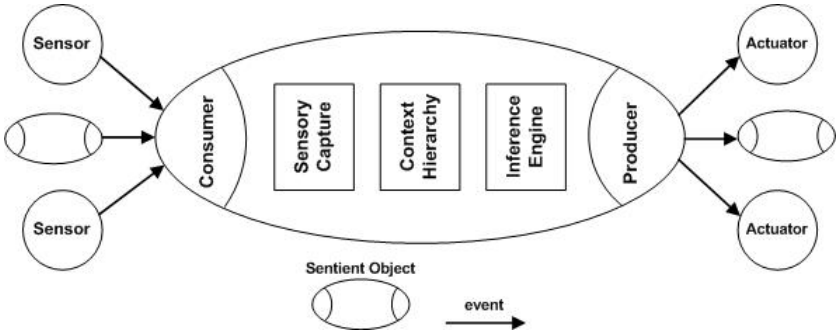


Figure 2.8: Sentient object architecture [12]

A middleware infrastructure which extends typical operating system concepts to include context-awareness was implemented in the Gaia project [84]. It aims at supporting the development and execution of portable applications for active spaces. The Gaia architecture is shown in (Figure 2.9).

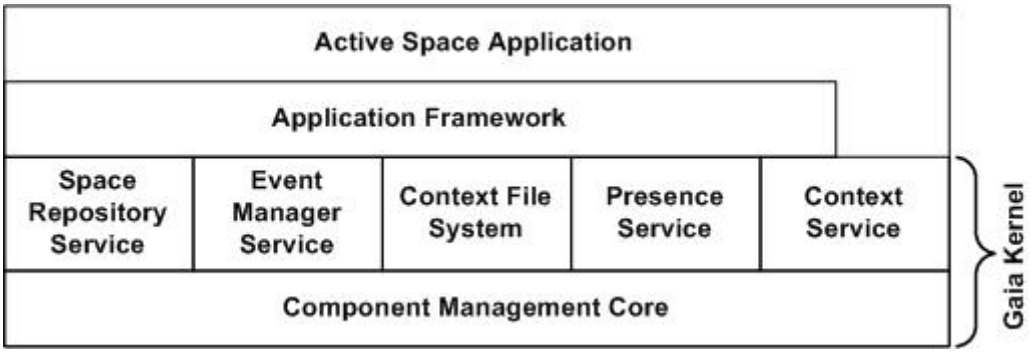


Figure 2.9: Gaia architecture [84]

2.2.4 Implementation

Generally, implementation is the realisation of an idea, model, design, specification, algorithm, etc. It is done by implementing the models's specifications of a context-aware system using a programming language (such as C++, Java). As a linguistic support to context-aware systems, [9] presented the Java Context-Awareness framework (JCAF). The goal of the JCAF is to provide a java-based, lightweight framework with an expressive, compact and small set of interfaces. The purpose of having such a simple and robust framework was to enable the programmers to extend it to more specialised context-awareness support in the creation of context-aware systems.

2.2.5 Testing

System testing is an investigatory testing phase, where the focus is to often have a destructive attitude. However,it does not test the design only, but also the behaviour of the system and even the expectations of the users.

2.2.6 Verification and Validation

Verification and validation are two important processes. They assure that each phase of building a system is going towards the right direction. To ensure the best results, these processes should be implemented between each two phases of building a context-aware system.

Verification is a quality process, used to evaluate whether or not a system or service complies with regulations, specifications or conditions imposed at the implementation phase. In other words, verification process is defined as (*Are we building the thing right?*).

Validation is the process of establishing evidence that provides a high degree of assurance that a system or service accomplishes its intended requirements. In other words, validation process is defined as (*Are we building the right thing?*). Validation process places a challenge to the context-aware systems [108], as existing techniques for validation such as model checking, testing and simulation can provide limited confidence in the correctness and efficiency of the application [67]. For instance, testing a context-aware system requires continuous feeding of contextual information, which is not easy to do. For that, greater confidence in a context-aware system can be gained if a legitimate validation technique is used throughout the process of developing that system.

2.3 Examples of Context-aware Systems

This section lists some examples of existed context-aware systems that have been built in the last decade.

2.3.1 Context-Aware Toolkit

Salber, Dey and Abowd [86] developed a context toolkit which relies on context widgets concept. They defined context widget to be a software component that provides applications with access to context information from their operating environment; just as the GUI context mediate between the application and the user.
Context widgets provide the following benefits [86]:

- They hide the complexity of the used sensors. For example, it hides the details of how a person's presence was sensed (Active badges, floor sensors or video image processing).
- They abstract context information to suit the expected needs of the system.

For example; a widget that tracks the location of a person within a building sends notifications to the application only when the person moves from one room to another, and ignores less significant moves.

- They provide reusable and customizable building blocks of context sensing. For example; a widget that used to track person's location can be also used in a variety of other applications, like office awareness or the tour guide. Furthermore a presence widget is used to sense persons' presence in a room, and a meeting widget may rely on a presence widget and assume the beginning of a meeting when two or more people are present in the same room.

They built two kinds of context widgets, the first one is the identity presence widget (Table 2.2), and the other is the activity widget. The function of the identity presence widget is reporting the identity and time of arrival and departure of people at a specified location. This provided information is useful for many applications like the tour guide. Generators are acquired in every widget. They are components that encapsulate sensor(s) and the software that acquires raw information from the sensor(s).

Table 2.2: Definition of the identity presence widget [86]

Widget Class	**Identity Presence**
Attributes	
Location	Location the widget is monitoring
Identity	ID of the last user sensed
Timestamp	Time of the last arrival
Callbacks	
PersonArrives(Location, ID, Timestamp)	Triggered when a user arrives
PersonLeaves(Location, ID, Timestamp)	Triggered when a user leaves

The activity widget (Table 2.3) senses the level of activity at a specific location. For example; it may be used to sense the presence of people in a room through their

activity like moving or speaking, so it can't sense the presence of people in a room by itself, but associated with other widget(s).

Table 2.3: Definition of the activity widget [86]

Widget Class	**Activity**
Attributes	
Location	Location the widget is monitoring
AverageLevel	Activity level (none, some, lot) averaged over a user-specified time interval
Timestamp	Time of the last activity level
Callbacks	
ActivityChange(Location, AverageLevel, Timestamp)	Triggered when the activity level changes from one level to another

2.3.1.1 Context-aware in/out board application (using context toolkit)

The first application which Salber, Dey and Abowd [86] built was a simple in/out board that is used in the offices commonly (Figure 2.10). The application is used to indicate the identity of the employees which are currently in the building and which are not, together with the time of their arrival or departure.

2.3.1.2 Context-aware information display application (using context toolkit)

The second application that they built is an information display application. The information displayed is very relevant to the user's location and identity, and changes to suit the user and the user's location and research group. The information appears on a display adjacent to the user. It is activated as a person approaches it.

The context information used is all about the location of the display, the identity of the user, the research group the users belong to and other needed information about the research group. In this case, a single widget is used which is the Group URL presence, this widget is placed near to the display, when a user presence is

Figure 2.10: A screen shot of the in/out board application [86]

sensed it sends a URL to the application and the application sends it to the nearby display. And again, here the application hides the details of how the user presence was sensed.

2.3.1.3 DUMMBO (using context toolkit)

For the third application, Salber et al. [86] wanted to augment an existed application which is DUMMBO (Dynamic Ubiquitous Mobile Meeting Board) project. DUMMBO used to record the meetings (i.e. records the ink written to and erased from the white board); the recording was initiated by writing or erasing activity on the physical white board. After the development made on DUMMBO, the recording is triggered when a group of two or more people gathered around the white board, and recording not only consisted of the ink written or erased, but the audio discussion also. All these details of the meeting can be then captured by a participant by indicating the date and the time of the meeting. Multiple Name presence widgets was used; one in each location where the DUMMBO could be moved to.

The above context toolkit was programmed using Java language (about 12400 lines of code) [86]. Four main points to talk in details about the context toolkit implementation which are: distribution, composition, heterogeneity and dynamism.

2.3.2 Context-Aware Personal System

A context-aware application has been developed by [99], in which they used personal sensors along with the environmental sensors. [99] has built their application according to the framework depicted in Figure 2.11, which relies on three key concepts:

- Context-aware applications require emphasising the components' autonomy.
- Two types of agents: context agent which captures the contextual knowledge and the personal agent which collects user's information.
- Agents are provided with ontology-based representation of data.

[99] defined four different levels in their architecture which are:

- Sensors: used to capture environmental information, deliver an electrical signal.
- Receiver: used to transmit the data from sensors to low level data, act as the first-level filter.
- Interpreter: at this point, the signal is translated into contextual data by using some modules like the HBR (Human Behavioural Recognition) module.
- Synthesizer: the most important level where the contextual data is translated into knowledge.

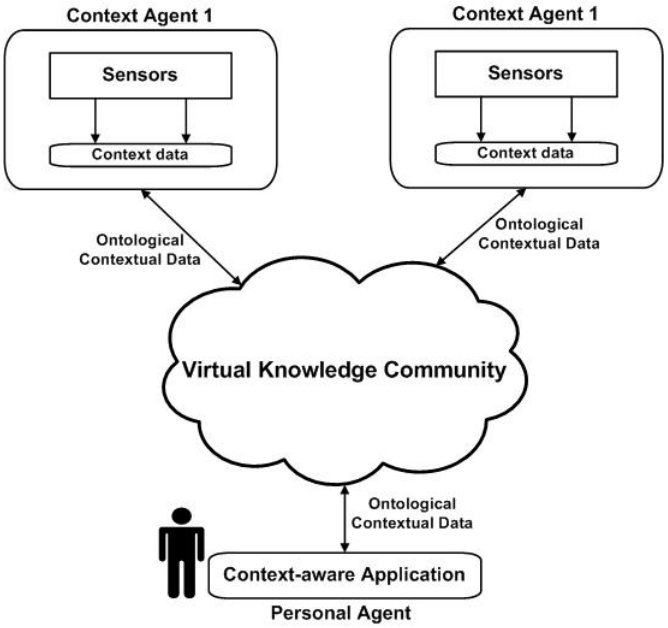


Figure 2.11: Description of the context-aware personal system framework [99]

The personal agent they designed is to be implemented on a wearable device, such as a hand-set phone or PDA. They implemented the personal agent in an application called 'Wake me up'. The scenario is as the following: a person takes the metro from any place to a known destination (station). The application warns the user for his/her station in order not to miss it through detecting his/her activity and to ring his cell phone if necessary. This scenario requires some devices: transmitter, activity sensor and cell phone.

This application requires that the metro station be supplied with a wireless transmitter in each station. The transmitter acts as a contextual agent that delivers information about the station. For the sensors, [99] used a foot pressure sensor (personal sensor) (Figure 2.12) to measure the activity level of the user. The measured activity level is one of the following activities: sleeping, sitting, standing, walking or running.

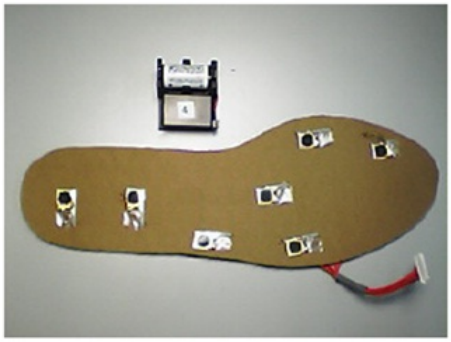


Figure 2.12: Foot pressure sensor [99]

For the personal agent, it was implemented on JADE-LEAP and was running on the Nokia N91. When the metro reaches the desired station, the personal agent detects the station's transmitter, extracts some information from it including the name of the station, if it is the desired one; then according to the activity level of the user, the suitable action will be taken.

2.3.3 Context-aware Health Care System

Kang *et al.* [56] proposed a wearable context-aware system for ubiquitous health-care. They designed two wearable sensor systems (Figure 2.13). The first one is a wrest type sensor system which has an ECG (ElecroCardioGram) sensor, a PPG (PhotoPlethysmoGraph) and a SKT (SkinTemprature) sensor. The other sensor type is a chest belt type sensor system which contains an ECG sensor, a respiration sensor, a SKT and a three axis acceleration sensor.

Figure 2.13: (a) Wrist type sensor system (b) Chest belt type sensor system [56]

They used a PDA as a wearable computer which contains a wireless LAN capability. Ubiquitous services are classified into two categories [56]: local services and remote services. As for the application, they made two applications; the first one is a Remote Health Monitoring Service which simply, transmits the collected signals from the wearable sensor to the doctors via the WWW service so they can monitor the user's health status.

The other application is the Self Health Check Service, which measures the bio-signals of the user via the wearable sensors and sends the results to the wearable computer after a series of processes so that the user can check his/her own health status.

2.3.4 Context-aware Mobile Phone

Schmidt *et al.* has integrated a sensors' board with a mobile phone to come up with a smart mobile phone that is aware of its context. Many mobile phones have the ability of switching their profiles automatically depending on a schedule. However, the profiles of the smart phone defines the way the mobile phone behaves when receives SMS, calls, etc.

The application layer exists in the mobile phone, thus, the work here is just to enhance the device with the ability of context awareness. The context-aware component was made small enough to fit with the battery pack of the Nokia 6110 mobile phone (Figure 2.14).

A profile of the smart mobile phone is selected automatically based on the recognised context. Table 2.4 lists the profiles which were defined in the mobile phone just for experimental purposes [93]. The user has a complete access to all these profiles and definitely can change any profile's action to what is desires.

Figure 2.14: The context-aware mobile phone with the sensor board [93]

Table 2.4: The defined profiles in the mobile phone (an example)

PROFILE	DESCRIPTION	ACTION
Hand	The user holds the mobile phone in his hand	Just vibrating
Table	Meeting Situation	Almost silent
Pocket	In a pocket, box, or suitcase	Loud ringing
Outside	The user is distant from the mobile	Loud ringing and Vibrating
General	None of the above	User setting

2.4 Overview on Policies

This section provides an introduction on what is a policy, how policies can be classified, and what are the terminology for policies. Also, it talks about security policies, what are the requirements for a security policy and what are the models of access control policies.

2.4.1 Policy

According to the online dictionary, a policy is simply a plan of action. [29] defined policies as rules governing the choices in behaviour of a system. In the same con-

text, a policy is a principle or rule to guide decisions and achieve rational outcome according to [2]. However, the Internet Engineering Task Force (IETF) organisation [3] defined a policy as a definite goal, course or method of action to guide and determine present and future work. The definition that satisfies the most has be proposed by [89] who defined a policy as:

> *A predetermined action pattern that is repeated by an entity whenever certain system conditions appear.*

It is not an easy task to define a policy [23] because of two reasons (i) the variety of applications in which policies are deployed (ii) the extensive number of available policy tools. There exist many types of policies, e.g. financial, commercial, military, security, social, etc. This classification is based on the related aspects to a policy, e.g. a military policy concerns mainly with *confidentiality*, while a commercial policy deals mainly with *integrity*.

2.4.2 Policy Terminology

There are more than one perspective to classify a policy. One is to classify policies according to their functionality within a system, or by their properties. The following has been proposed by [23]:

2.4.2.1 Policies as Rules Dictating Behaviour

It is the most common used type of policies. This type of rules are *procedural* in its nature, i.e. it indicates what specific action should be taken under special circumstances. These rules dictate the behaviour of the system. For example:

- Upon system infection, if the virus is removed, then do a system's back up.
- At 5:00 p.m everyday, shut down the computer.

The first example declares that if the system is infected, a condition should be checked (if virus is removed), and if TRUE then perform the action (system's back up). The second policy example, a specific action (system shut down) must be performed at a specific time (5:00 p.m). However, both of the above examples tell the story of what has to be done when a specific condition is triggered.

2.4.2.2 Policies as Rules Granting or Denying Permissions

This type of policies also comes in the form of rules, but rather than being procedural, they are *declarative*. They are used to describe whether to grant or deny permissions to entities to perform a specific action. For example:

- Allow user X to access management files.

2.4.2.3 Policies as Constraints or Parameters

Policies are often naturally used to describe the constraints or parameters that rule a system. These constraints are specified in a *declarative* manner; they provide declaration statements about specific relationships that must holds TRUE in a system at all times. These constraints are also called policies because they dictate rules of the system. Such policies are used to provide high-level targets and objectives to achieve. For example, in military, such constraints comes in the form of orders from a number of different resources such as: organisational directives, operational guidance, administration manuals or orders from higher headquarters. This form of policies may not be directly understandable by the system's management, thus it may be needed to translate those policies into a system usable structure. An examples on a policy acting as a constraint:

- Network traffic should be monitored every 60 seconds.

2.4.3 A Taxonomy of Policies

This section talks about the types of policies in terms of their functionality.

2.4.3.1 Security Policies

Security policy defines the condition of an authenticated subject (user) is allowed to execute a specific action on an object (target) [54]. According to *Bishop* [14], a *security policy* is a statement that partitions the states of the system into a set of *authorised*, or *secure* states and a set of *unauthorised*, or *nonsecure* states. A security policy sets the context in which a secure system can be defined. Mainly there are two types of security policies:

- **Military security policy:** is a security policy developed primarily to provide confidentiality.
- **Commercial security policy:** is a security policy developed primarily to provide integrity.

2.4.3.2 Access Control Policies

Access Control (AC) policies are considered as a type of security policy for they concern with controlling the access to a system, thus providing more security. A security policy may have two types of access control, alone or in combination [14]. The owner of an object is discrete in managing the access control of this object, on the other hand, the operating system controls access of a certain object, in which case the owner can not override the controls.

Models of Access Control Policies Modelling of access control policies has been divided mainly into: *discretionary access control* (DAC), *mandatory access control*

(MAC) and *Role-Based Access Control* (RBAC). In DAC, the owner of an object can decide whom to grant access rights to depending on the identity of the subjects and objects, also it concerns with controlling subjects access to objects. In MAC, a central authority decides whom to have access rights based on fixed regulations, it concern with controlling the flow of information between different objects in a system [94]. In addition, RBAC controls subjects access to objects based on the roles users have within the system. RBAC share DAC in having an administrative policy defining who can be granted access rights to certain objects.

Types of Access Control Policies AC policies can be divided into several types, such as:

- *Authorisation policy:* define what activities a subject can perform on a set of objects. This type of policies represent the essence of access control policies. A *positive* authorisation policy defines the actions that a subject is allowed to perform objects. A *negative* authorisation policy defines the actions that a subject is *not* allowed to perform on objects.

- *Delegation policy:* define the condition of a subject can delegate a specific access right to another subject.

- *Obligation policy:* define the condition in which a subject must or must not do a set of actions on specific objects.

- *Refrain policy:* define the actions that a subject must refrain from performing (must not perform) on objects even though it may be permitted to perform those actions.

2.4.3.3 Management Policy

A management policy is used to specify dynamic adaptable management strategies, so they can be easily modified to change the management approach without modifying the management system itself. Most of the policy-based management approaches use condition-action rules (aka if-then rules) for they can be easily modified as they simply contains a *condition* and an *action* [28]. The most notable piece of work in this area is the policy model proposed by the Internet Engineering Task Force (IETF), the rule is described as follows:

```
if <condition(s)> then <action(s)>
```

The condition-part of the rule can be described in a simple disjunctive normal form or compound conjunctive expression. The action-part of the rule can be a single or set of actions that must be performed when the conditions are satisfied. This type of policy rule prescribes similar behaviour to an obligation policy, however, there is no explicit event specification to trigger the execution of the actions, but instead, it is assumed that an implicit event such as a user request will trigger the policy rule as a whole.

2.4.4 Conflicts in Policies

Some policy-based management systems deploy complex sets of policies. Due to the dynamic nature of some of these systems; policies may change, leading to conflicts in the deployed policies. Conflict in policies means that actions of some policies override actions of others, this is due to the overlapping of conditions of some policies. One way to solve conflict in policy rules is to associate them with a priority value to resolve conflicts between rules [66]. However, this approach is not scalable in large systems with a large number of rules.

2.4.5 The Event-Condition-Action (ECA) Model

Some researchers deal with the ECA as a type of policy as it simply contains a condition and action. In general, the ECA is a simple pattern used to represent policies. Policies represented by ECA rules perform actions automatically in response to events provided that stated conditions hold. Such a simple representation allows the reactive functionality of a system to be specified and managed within a single rule base, thus enhancing the modularity and maintainability of the system [78]. An ECA rule has the general syntax of:

`on` *event* `if` *condition* `do` *actions*

The ECA model contains the following components:

- *Events:* provide means of triggering policies, events represent incidents which happen asynchronously in the relevant domain (i.e. environment) of a system. Events are optional components of policies.
- *Conditions:* are used to check whether a specific rule's condition expression is set to TRUE, so a specific action can be taken. Conditions are optional components of policies.
- *Actions:* operations that are performed on a specific event and/or after a policy condition is set to TRUE. An action can be used to trigger another policy's condition. Actions are compulsory component of a policy.

Executing a rule's action may be the trigger to further ECA rules and the rule execution proceeds until no more policy rules are triggered. However, this may lead to non-termination of rule execution process, thus much research has focused on the development of static rule analysis techniques for detecting possibly non-terminating

rule sets [82].

Almost any policy rule can be represented using the ECA model, and especially *management policies*. Management policies usually contain simple rules (if-then), thus defined and represented in means of an ECA pattern. For example, assume a policy stating that '*Upon system infection, if the virus is removed, do system's back up*'. Using ECA model, this policy can be represented as follows: *on* system infection, *if* virus removed, *do* back up. Where the event that triggers the policy rule is the *system infection* part, and the state or condition to be evaluated is that *if the virus has been removed*, and if the condition holds to *true* then the action to perform *system back-up* will be done.

2.5 Summary

In this chapter, we have talked about context, why using context and how to capture it. Then we defined context-aware systems extensively and illustrated a table which contains some examples on context-aware systems. We then talked about the development process of a context-aware system and its phases and identified some of the challenges lies within the development process and what is the work that have been done so far. Moreover, we gave some examples on projects that have been done with context-awareness in mind.

In the second part of our literature review, we talked about policies. We defined what is a policy in general and what is the policy terminology. Then, we gave a taxonomy of policies in terms of their functionality. Finally, we presented a simple policy representation model and identified its components.

The next chapter (Chapter 3) talks about the computational model used in the work of this thesis. the computational model consists of an ambient-based model in addition to the mathematical notation of `CCA`.

Chapter 3

Computational Model

Objectives:

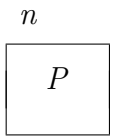
- Define an ambient-based computational model
- Present the characteristics of an ambient
- Present the mathematical notation of `CCA`
  - modelling in `CCA`
  - syntax of `CCA`
  - semantics of `CCA`
- Show how policies can be specified in `CCA`

3.1 Computational Model

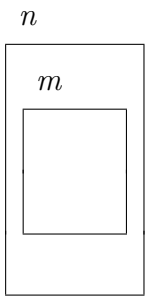
In this section, we define a computational model on which our work throughout the thesis will be based. Our computational model is based on the notion of ambient which already exists and has been used in formal modelling such as: Mobile Ambients. An ambient is an entity used to describe any object or component (e.g. person, process, device, location, etc.) in a system [22]. An ambient is defined as:

Ambient: is a bounded place where computations happen. It has a name, a boundary, and can have other ambients inside it. An ambient can be mobile and has the ability to communicate with other ambients.

An ambient is represented graphically as follow:

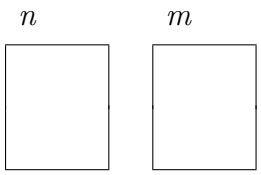


where n is the name of the ambient, and P is the name of a process which represents the behaviour of ambient n. An ambient can have other ambients inside its boundaries, allowing a set of ambients to be represented hierarchically. For example, ambient n can have ambient m inside.

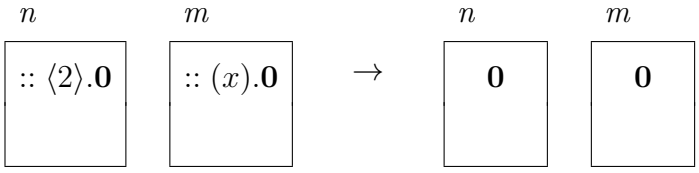


There are three possible relations for an ambient to have with other ambients;

parent, child and sibling. Ambient x is a parent to ambient y if y is inside x. In the same manner, ambient y is considered as a child to ambient x. Ambient x is a sibling to ambient y, if x and y share the same parent. For example, ambient n is a parent to ambient m, as ambient m is a child to ambient n, as depicted in the above graph. Ambient n is a sibling to ambient m, and can be represented as follows:

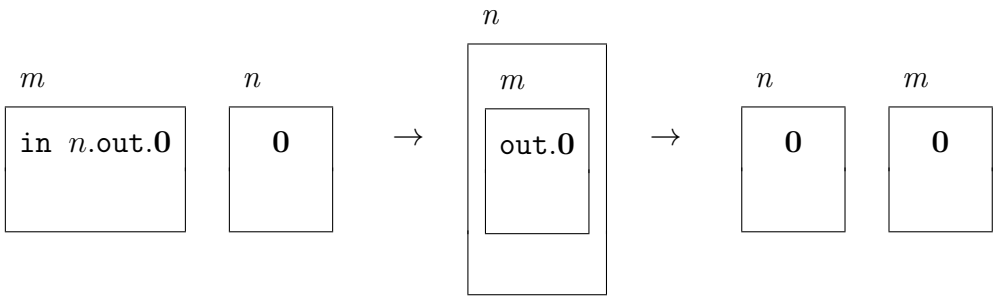


Since the process calculus of `CCA` will be presented in the next section, we are going to borrow some `CCA` capabilities of an ambient to use in this section. An ambient is capable of communicating with other ambients around it (i.e. parent, child and sibling ambients). Ambients can send/receive messages between each other. The process of exchanging messages is done by using the handshaking process. For example, if an ambient, say n, sends a message, say 2, to an ambient, say m, in its turn, ambient m has to be able to receive such a message from ambient n to complete the process of exchanging a message (handshaking). This is represented as follows, where :: is the symbol of a sibling ambient, $\langle\rangle$ means sending and () means receiving, and $\mathbf{0}$ is a process that does nothing and terminates immediately. After exchanging the message, the two ambients will terminate.



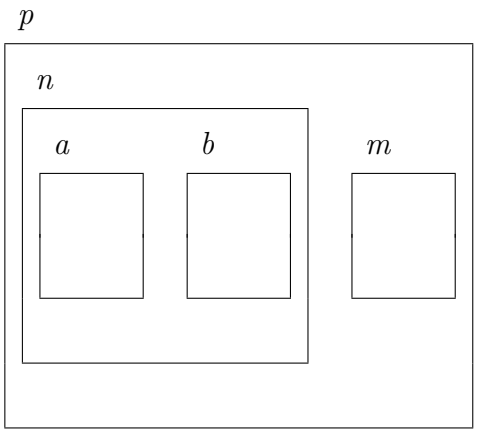
However, an ambient can also perform the procedure of handshaking with a parent ambient or a child ambient, using the symbols $\uparrow$ and $\downarrow$, respectively.

In terms of mobility, an ambient has the ability to move around in its environment. An ambient can move into a sibling ambient, and out of its parent ambient only. However, as a default process, if an ambient move around its environment, then all the child ambients of this ambient move with it. In `CCA`, an ambient has two mobility capabilities; `in` and `out`. An ambient can perform the capability `in` x to move into the ambient x, or the capability `out` to move out of its current parent. For example, ambient m can move into ambient n, and then move out again, as follows:



In terms of context-awareness, an ambient is aware of its surrounding environment. It has the ability to sense the context around it (i.e. other ambients). Parent ambient, child ambients, sibling ambients and other ambients around a specific ambient identify the environment of that ambient. The location of an ambient is its parent ambient, we say n is the location of ambient m since ambient m is inside ambient n. For example, the environment of the ambient m in the graph below

includes the ambients p, n, a and b.



Now that we have presented what an ambient is, and what are the characteristics that give it the potential to model any entity in a system, in the next section, we present CCA, which is a process calculus used to model the behaviour of ambients.

3.2 The Calculus of Context-aware Ambients (CCA)

Pervasive computing [104] is the new paradigm applied to make computers integrated (but behind the scenes) into everyday life activities. What pervasive computing adds to the concept of distributed systems is that applications support mobility and context-awareness. In terms of mobility, entities (i.e. users and devices) within the system can be mobile. For a system to be context-aware, it is required that it has the ability to sense the context within its environment, and use the information to adapt its behaviour to the current situation.

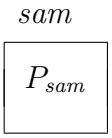
The Calculus of Context-aware Ambients (CCA) [95, 96] has been proposed to model such applications. CCA is built upon the calculus of Mobile Ambients (MA in short) [22] and introduces new constructs to enable ambients (such as users, devices, locations, software, etc.) and processes to have a sense of awareness of the

environment they are being executed in. The result is a powerful process calculus where mobility and context-awareness are first-class citizens.

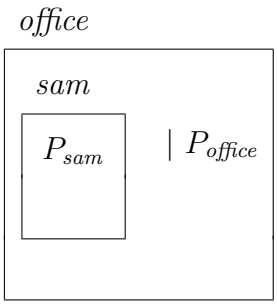
3.2.1 Modelling in CCA

The Calculus of Context-aware Ambients (CCA) is a process calculus based on the notion of ambients. CCA is used to model ambients in terms of process, location and capability. As discussed in the previous section, an ambient can be modelled graphically, as a set of ambients. So, in this section, we will demonstrate how to model a simple example of a user and its surrounding environment. The process of modelling an ambient, its context, its mobility and capabilities will be in the shadows of modelling our example.

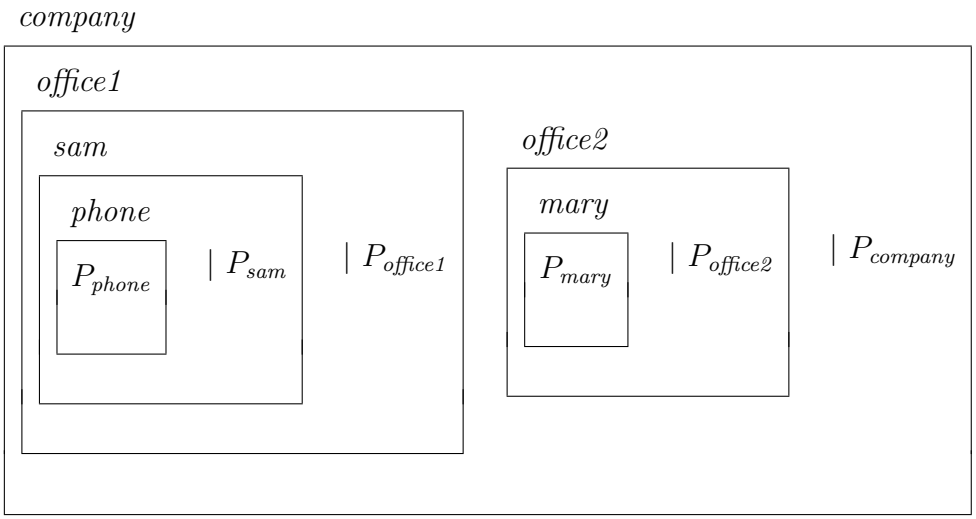
Therefore, as a first step to demonstrate our example, suppose a user, say Sam, can be modelled as follows, where P_{sam} is a process specifying the behaviour of the ambient/user Sam.



Any entity and object has a location in real life; so has the model we use. CCA based modelling can model a location for every ambient. The location of an ambient is modelled as a parent ambient. For example, the location of the user Sam can be modelled as follows, where the ambient *office* represents the location of the ambient Sam:



Any ambient is aware of its context/environment. The environment of an ambient is defined by other ambients around it. The context for the user Sam is represented by the user's location, other users, locations around it, available resources, etc, where the user Sam is carrying a Phone.

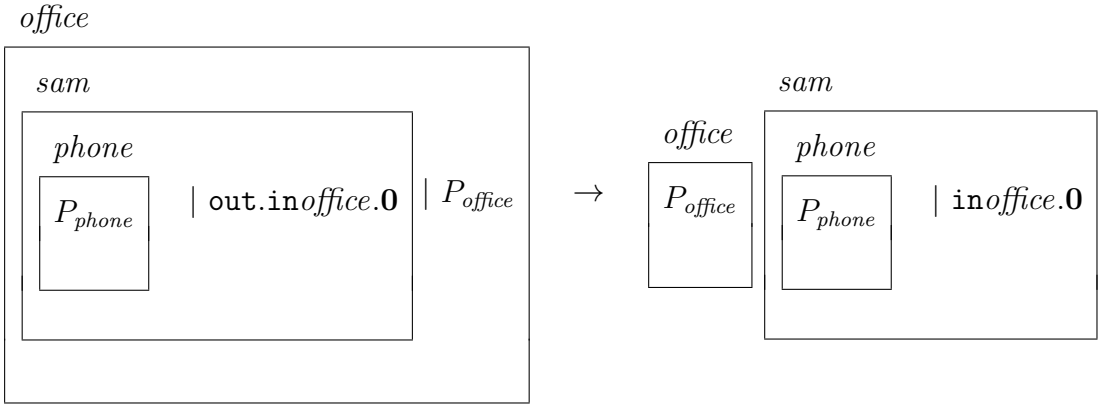


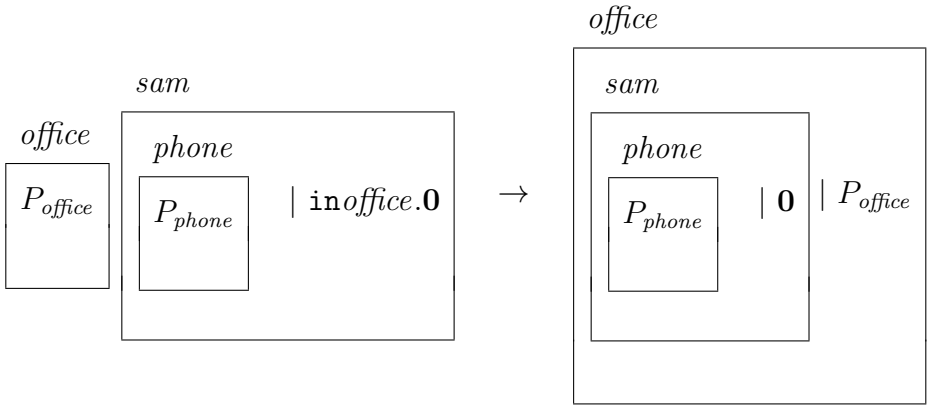
Where the environment for the user Sam is represented by every ambient around that user (company, office1, phone, office2 and Mary), in which Sam is a part of the ambients forming the environment. The above graph can be represented textually as follows:

$$\begin{aligned} company \quad & [\ office1[\ sam[\ phone[P_{phone}] \mid P_{sam}] \mid P_{office1}] \\ & \mid office2[\ mary[\ P_{mary}] \mid P_{office2}] \\ & \mid P_{company}\ ] \end{aligned}$$

An ambient can be mobile; it can move around within its environment. In `CCA`, `in` and `out` are two essential mobility capabilities that enables ambients to move out from one location (i.e. parent ambient) to another. The capability `in` is used by an ambient to move inside a sibling ambient. An ambient can use the capability `out` to move out of its parent ambient. As ambients can be represented hierarchically, when

an ambient moves around, all its child ambients move with it. For example, the user *Sam* can move out of *office1* and then move in again by performing the capabilities `out` and `in` respectively, noting that the *phone* ambient is a child ambient to Sam on all occasions, as follows:



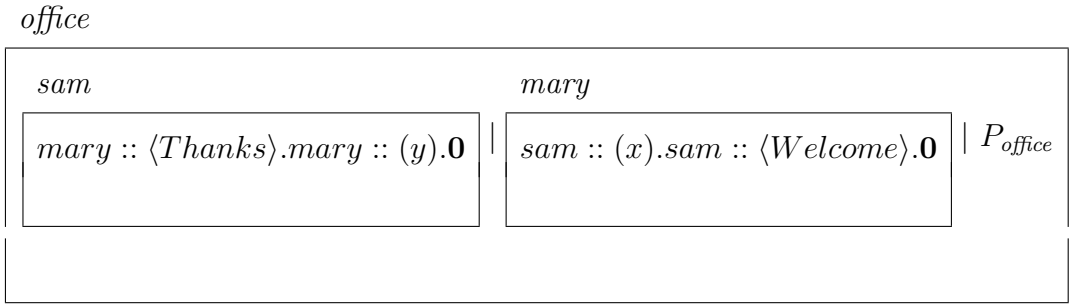


An ambient can communicate with other ambients around it. As mentioned earlier, communication between the ambients is performed using the handshaking process. There are two possible scenarios for an ambient to communicate with other ambients (in terms of their location):

- parent-child communication: where a parent ambient and a child ambient exchange messages (using the location symbols ↑for a parent and↓for a child)

- sibling-sibling communication: where two sibling ambients exchange messages (using the location symbol :: for a sibling)

For example, if user Sam presses a button on the phone device he is carrying, this is a parent-child communication, while if he chats with a sibling ambient (i.e. both have the same parent ambient), then it is a sibling-sibling communication. Now to demonstrate an example on the simple handshaking communication process between ambients, suppose the user Sam is with user Mary in the same office, and when Sam says 'Thanks', Mary responds with a 'Welcome' message, as follows:



An ambient is context-aware if it can sense the context in its environment and react to the information obtained from that context. A simple, yet powerful, capability exists in `CCA` which gives an ambient the sense of context-awareness: it is the context-guarded prefix [1]. It has the form of $\kappa?M.P$, where κ represents a context expression, M is a capability and P is a process. This context-guarded prefix is used to model context-awareness. It means that the capability M and the process M will not be operated unless the context expression κ is met first (i.e. the `CCA` expression $\kappa?M.P$ is equivalent to the normal expression of *if-then*).

For example, suppose users Sam and Mary are working in the same office, and if Sam is using his mobile phone and Mary is in the office, then Sam should go out of the office. This capability is modelled as follows, where $\texttt{at}(n)$ is a context

[1] This will be explained further in a forthcoming section

expression that holds if the parent ambient of the ambient evaluating that context is n, `onPhone` is an abstract context expression which is true if user Sam is using his mobile phone, `with(`n`)` is a context expression that holds if the ambient evaluating that context is a sibling of ambient n, and '$\wedge$' refers to the logical operator *and*.

sam

$$\boxed{(\texttt{at}(\mathit{office}) \ \wedge \ (\mathit{onPhone}) \ \wedge \ \texttt{with}(\mathit{mary}))?\ \mathit{out}.\mathbf{0}}$$

Up to this point, we have demonstrated one simple example through which we explained the main characteristics of an ambient. The next section explores the formalism in `CCA`, giving the syntax of `CCA`, its semantics and some examples for illustration.

3.2.2 Syntax of `CCA`

This section presents the syntax of processes and capabilities of the `CCA` notation. Just like in the π-calculus [70, 88], the simplest entities that form the notation are *names*. The names are used to name the ambients, where ambients are users, locations, or devices. To give the syntax of `CCA`, we need to assume a set of names, to be used where appropriate, which are written in a lower-case letters, e.g. n, x and y. We let $\tilde{y}$ denote a list of names and $|\tilde{y}|$ the arity of such a list. As depicted in Tables (3.1, 3.2, 3.3, 3.6), four syntactic categories are distinguished: processes P, capabilities M, locations α and context expressions κ (read *kappa*).

3.2.2.1 Processes

The process $\mathbf{0}$ does nothing and terminates immediately. The process $P|Q$ indicates that process P is operating in parallel with process Q. The process $(\nu n)\ P$ states that the scope of the name n is limited to the process P. The sign of replication is

!, so the replication process $!P$ signifies a process which can create a new replica of the process P whenever needed, i.e. $!P \equiv P|!P$. The process $n[P]$ designates an ambient named n where P is the process that describes its behaviours. The pair of square brackets '[' and ']' outlines the boundary of that ambient, so everything that lies between those brackets are part of the ambient's behaviour.

Table 3.1: Syntax of `CCA`: processes

P, Q	Description
$\mathbf{0}$	inactivity
$P\|Q$	parallel composition
$(\nu n)\ P$	name restriction
$n[P]$	an ambient
$!P$	repetition
$\kappa?M.P$	context-guarded capability
$x \triangleright (\tilde{y}).P$	process abstraction

A context expression specifies the condition that must be matched by the environment of the executing process. The *context-guarded prefix* $\kappa?M.P$ is a process that waits and does not perform the capability M followed by the process P until the surrounding environment satisfies the condition which is the context expression κ. The context-guarded prefix is much like the *If-Then* statement in its functionality, because the If condition Then action statement also perform the action if the condition(s) is/are met. The use of context-guarded prefix is one of the two main mechanisms for context acquisition in `CCA` (the second mechanism for context acquisition is the call to a process abstraction as discussed below).

Finally, the process abstraction $x \triangleright (\tilde{y}).P$ denotes the linking of the name x to the process P, where $\tilde{y}$ is a list of *formal parameters*. The linking is local to the executing ambient in which the process abstraction is defined. So a name x can be linked to more than one process in different ambients. For example, name x is

linked to process P in ambient n and to process Q in ambient m. A call to a process abstraction named x is done by the executing ambient by performing the capability $\alpha\ x\langle\tilde{z}\rangle$ where α specifies the location in which the process abstraction is defined and $\tilde{z}$ is the list of *actual parameters*.

3.2.2.2 Location

As ambients can communicate between each other, the location is an important parameter to be used as a reference by the communicating ambients. An ambient can exchange messages with a parent ambient, child ambient or sibling ambient. So, the location α can be '$\uparrow$' which denotes any parent, '$n\uparrow$' for a specific parent ambient named n, '$\downarrow$' which refers to any child, and '$n\downarrow$' for a specific child ambient named n, the '::' symbol is used to signify any sibling, and 'n ::' indicates a specific sibling ambient named n. However, an ambient can use ϵ (empty string) to refer to itself (calling ambient).

Table 3.2: Syntax of `CCA`: location

α	Description
$\uparrow$	any parent
$n\uparrow$	parent n
$\downarrow$	any child
$n\downarrow$	child n
::	any sibling
n ::	sibling n
ϵ	local

3.2.2.3 Capabilities

There are two mobility capabilities which enable an ambient to move around in its environment, `in` and `out`, defined in `MA` [22]. An ambient can perform the capability `in` n to move into a *sibling* ambient named n, and the capability `out` enables an ambient to move out of its current parent ambient.

Table 3.3: Syntax of `CCA`: capabilities

M	Description
`in` n	move into ambient n
`out`	move out
$\alpha\ x\langle\tilde{y}\rangle$	process call
$\alpha\ (\tilde{y})$	input
$\alpha\ \langle\tilde{y}\rangle$	output
`del` n	delete ambient n

A process call $\alpha\ x\langle\tilde{y}\rangle$ behaves like the process linked to x at location α, in which each actual parameter in $\tilde{y}$ is substituted for each occurrence of the corresponding formal parameter. A process call can only take place if the corresponding process abstraction is *available* at the specified location. Ambients can exchange messages (send and receive). Using the capability $\alpha\ \langle\tilde{y}\rangle$ an ambient can send a list of names $\tilde{y}$ to a location α. To receive, an ambient can perform the capability $\alpha\ (\tilde{y})$ to receive in the variables in $\tilde{y}$ a list of names from a location α.

The capability `del` n deletes an ambient named n, only if that ambient is of the form $n[\mathbf{0}]$, i.e. empty/ineffectual ambient. Ambient n must be situated at the same level as that capability, i.e. the ambient performing that capability is the parent ambient of n. For example, the process `del` $n.P \mid n[\mathbf{0}]$ reduces to P. The capability `del` is considered as a garbage collector that deletes empty ambients which have no more computations to execute, i.e. $n[\mathbf{0}]$.

Example 3.2.1

1. *The capability* $\uparrow x\langle\tilde{z}\rangle$ *calls the abstraction* x *defined of the calling ambient;*
2. *The capability* $::\ x\langle\tilde{z}\rangle$ *calls the process abstraction* x *defined in one of the siblings of the calling ambient;*

3. *The capability $n{\downarrow}x\langle\tilde{z}\rangle$ calls the process abstraction x defined in a child ambient named n of the calling ambient;*

4. *The capability $x\langle\tilde{z}\rangle$ calls the process abstraction x defined in the calling ambient.*

3.2.3 Context model

Like Mobile Ambients (MA), the basic structure to represent any entity in any system in `CCA` is the ambient. As mentioned before, an ambient has a name, a boundary and can host and be nested inside other ambients, which allows a set of ambients to be represented hierarchically. A hole, which is denoted by $\odot$, is a spatial context that represents the position of the executing process in the system. For example, suppose a system is modelled by the process $P \mid n[Q \mid m[R \mid S]]$. So, the context of the process R in that system is $P \mid n[Q \mid m[\odot \mid S]]$, and that of the ambient named m is $P \mid n[Q \mid \odot]$, and that of ambient n is $P \mid n[\odot]$. Thus the contexts of `CCA` processes are described by the grammar in Table 3.4, and a property of a context is called context expression (CE in short).

Table 3.4: Syntax of contexts

E	Description
$\mathbf{0}$	nil
$\odot$	hole
$n[E]$	location
$E\|P$	parallel composition
$(\nu n)\ E$	restriction

In Table 3.4, E represents the context, n stands for names and P ranges over processes [2]. The context $\mathbf{0}$ represents an empty context, in which it does not provide any information. The hold $\odot$ indicates the position of a process in that process's context. The context $n[E]$ signifies that the context E is the internal context of the

[2]See Table 3.1

ambient n. The context $P|E$ indicates that the process P operates in parallel with the context E; in other words, the context E is part of the context of process P. The context $(\nu n)\ E$ refers to a name restriction, in which the name n is restricted to the context E.

Example 3.2.2

The context of the mobile phone carried by user Sam while he is with friend Mary at the office is represented as follows, where P models the remaining part of the office's internal context and Q models the internal context of the user Mary.

$$e_1 \ \widehat{=}\ office[P \mid sam[\odot] \mid mary[Q]]$$

Example 3.2.3

Assume the user Sam went out of the office; then, the context of the mobile phone carried by Sam is described as follows:

$$e_2 \ \widehat{=}\ sam[\odot] \mid office[P \mid mary[Q]]$$

Example 3.2.4

User Sam might hold another device with him, a PDA say, so the context of the smart mobile phone while carried by Sam, who is inside the office, can be modelled as follows:

$$e_3 \ \widehat{=}\ office[P \mid sam[\odot \mid pda[R]]]$$

where pda is the name of the ambient modelling the PDA device carried by Sam, and R is a process specifying the functionality of the PDA.

Table 3.5 gives the algebraic semantics of contexts in terms of equality. The first equality (cont-0) indicates that a context with a value of **0** is a nil context, i.e. would not affect other contexts running in parallel with it. (cont-1) says that

parallel compositions of context are commutative, while (cont-2) says that they are associative. The equality (cont-7) states that equality propagates across scope, (cont-8) indicates that equality also spreads across parallel composition and (cont-9) says that equality spreads through ambient nesting as well.

Table 3.5: Algebraic semantics of contexts

$E \mid \mathbf{0}$	$=$	E	(cont-0)
$E_1 \mid E_2$	$=$	$E_2 \mid E_1$	(cont-1)
$E_1 \mid (E_2 \mid E_3)$	$=$	$(E_1 \mid E_2) \mid E_3$	(cont-2)
$(\nu n)(\nu m)\ E$	$=$	$(\nu m)(\nu n)\ E$	(cont-3)
$(\nu n)\ E_1 \mid E_2$	$=$	$(\nu n)\ (E_1 \mid E_2)$ if $n \notin \mathbf{fn}(E_2)$	(cont-4)
$(\nu n)\ m[E]$	$=$	$m[(\nu n)\ E]$ if $n \neq m$	(cont-5)
$(\nu n)\ \mathbf{0}$	$=$	$\mathbf{0}$	(cont-6)
$E_1 = E_2$	$\Rightarrow$	$(\nu n)\ E_1 = (\nu n)\ E_2$	(cont-7)
$E_1 = E_2$	$\Rightarrow$	$E_1 \mid E_3 = E_2 \mid E_3$	(cont-8)
$E_1 = E_2$	$\Rightarrow$	$n[E_1] = n[E_2]$	(cont-9)

3.2.4 Context Expressions

Table 3.6 presents the Context Expressions (CEs in short). The CE **True** always holds, where it indicates the value *true*. The CE $\bullet$ holds for the hole context only, i.e. it represents the process evaluating that context expression. For the CE $n = m$, it applies only if the names n and m match exactly, i.e. are lexically identical. Such CE is important for checking whether two messages have the same content or not.

The first order operators $\neg$ (negation), $\wedge$ (and) and $\exists$ (there exist) expand their standard meaning to CEs.

A CE $\kappa_1|\kappa_2$ holds for a context if that context is a parallel composition of two contexts such that κ_1 holds for one and κ_2 holds for the other. A CE $n[\kappa]$ holds for a context if that context is an ambient named n such that κ holds inside that

ambient. A CE $\texttt{new}(n, \kappa)$ holds for a context if that context is a restriction of the name n to another context for which κ holds. A CE $\oplus\kappa$ holds for a context if that context has a child context for which κ holds. A CE $\varpropto\kappa$ holds for a context if there exists somewhere in that context a sub-context for which κ holds. The operator $\varpropto$ is called *somewhere modality* while $\oplus$ is aka *spatial next modality*.

Table 3.6: Syntax of `CCA`: context expressions

κ	Description
True	true
$\bullet$	hole
$n = m$	name match
$\neg\kappa$	negation
$\kappa_1 \vert \kappa_2$	parallel composition
$\kappa_1 \wedge \kappa_2$	conjunction
$\texttt{new}(n, \kappa)$	revelation
$\oplus\kappa$	spatial next modality
$\varpropto\kappa$	somewhere modality
$\exists x.\kappa$	existential quantification

There are some derivatives that can be derived logically, such as **False** $\widehat{=}$ $\neg$**True**, where both sides implies the value *false*. The disjunction relation $\kappa_1 \vee \kappa_2$ can be expressed logically as $\neg(\neg\kappa_1 \wedge \neg\kappa_2)$. The implication of $\kappa_1 \Rightarrow \kappa_2$ is also equivalent to $\neg\kappa_1 \vee \kappa_2$. Finally, a logical equivalence of two contexts $\kappa_1 \Leftrightarrow \kappa_2$ is equivalent to $(\kappa_1 \Rightarrow \kappa_2) \wedge (\kappa_2 \Rightarrow \kappa_1)$.

See Appendix A for some examples of predicates that can be used to specify common context properties such as the location of the user, with whom the user is and what resources are nearby.

3.2.5 Data Structures

This section presents two types of data structures; one-place buffer and persistent cell. It shows how such data structures can be modelled in a natural fashion using

CCA.

3.2.5.1 One-Place Buffer

A one-place buffer is a data structure that can be activated using two actions; *push* and *pull*, hence the name; a buffer can store only one item at a time. So, it is not possible to push an item into a full buffer nor to pull an item out from an empty one. Then, the action push can be performed to an empty buffer, and the action pull can be performed to a full buffer only.

Initially, we assume a buffer is empty, so only a push action can be performed. After an item is pushed into the buffer, only a pull operation can be performed. The buffer is modelled as an ambient named *buf* as follows; it waits for its parent to perform a *push* action, and after that, it waits for its parent to perform a *pull* action, then terminates.

$$buf \left[\begin{array}{l} \uparrow(x).\langle x\rangle.\ \mathbf{0} \\ \mid\ !(y).\uparrow\langle y\rangle.\ \ \uparrow(z).\langle z\rangle.\ \mathbf{0} \end{array} \right]$$

This ambient waits for its parent to perform a *push*, and once a push action has been performed the ambient waits for its parent to perform a *pull* action. These actions are specified by the following processes:

$$push(v) \mathrel{\widehat{=}} buf\downarrow\langle v\rangle$$
$$pull(x) \mathrel{\widehat{=}} buf\downarrow(x)$$

3.2.5.2 Persistent Cell

A persistent cell models a data structure that works by two operations *put* and *get*. Like the buffer, a persistent cell can store only one item at a time, but unlike a

buffer, it keeps the item and gives only a copy of it when requested. So, when a *put* action is performed, the cell stores the item. The cell then keeps its content after a *get* action is performed. However, when a *put* action is performed to a full cell, it replaces the item in the cell with a new one. Because a persistent cell can keep its content after performing the *pull* action, it is used more often with regards to the buffer.

A persistent cell can be modelled by the following ambient, where 5 is the initial value in the cell:

$$cell \left[\begin{array}{ll} & \langle 5 \rangle.\mathbf{0} \\ | & !\uparrow().(w).(\langle w \rangle \mid \uparrow\langle w \rangle).\mathbf{0} \\ | & !\uparrow(x).(y).(\langle x \rangle \mid \uparrow\langle\rangle).\mathbf{0} \end{array} \right] \tag{3.1}$$

A *get* action is performed as follows, where y is a variable that will receive the value stored in the cell and may occur free in the continuation process P:

$$cell\downarrow\langle\rangle.\mathbf{0} \mid cell\downarrow(y).P \mid \text{Eq. (3.1)}.$$

Similarly, a *put* action is performed as follows, where 7 is the value to put in the cell and Q is a continuation process:

$$cell\downarrow\langle 7\rangle.\mathbf{0} \mid cell\downarrow().Q \mid \text{Eq. (3.1)}.$$

3.3 Policy Specification

In general, a policy is a set of rules used to govern the behaviour of a system or an entity within a system. In this section, we talk about how policies can be specified in `CCA`. Then we will introduce a context-aware policy, followed by its specification.

3.3.1 Policy Representation

There are many types of policies, such as security, commercial, military, privacy, etc. The type of a policy concerns the aspect in which a policy is interested, or the purpose for which a policy is applied. There are different names, functions and classifications for policies; however, the common feature among policies is that, they are all rules applied to control/manage a system. The classification in which we are interested, is the policy terminology, which classifies the policies according to the way they behave. A policy can be set to control the access to management files of a system, to control the access to a military base, to set a constraint, to configure system behaviour on special occasions / events, to govern the traffic within a network, etc. Accordingly, policies in general can be classified in terms of behaviour into the following:

- rules dictating behaviour;
- rules granting or denying permissions; or
- constraints

Policies can be represented using the traditional conceptual model '*Event-Condition-Action*' (ECA). The ECA model is represented by three parameters: on *event*, if *condition* is satisfied, then perform *action*. The ECA model operates as follows: a system monitors previously specified events, these events then are used to trigger the evaluation of one or more policy rule's conditions. If an event triggers a condition that matches the policy rule, then one or more policy actions specified by the policy rule are executed [27]. An ECA rule has the form:

ON event

IF condition

THEN action

The ECA model is a representation of the simple *if-then* statement. However, complex policies can also be specified by combining two or more policies together, where a condition may represent a set of conditions; also, an action may represent a set of actions, and one policy action could be the trigger to another policy condition. Figure 3.1 below depicts a block diagram representing the ECA policy model.

Figure 3.1: Block diagram: ECA policy model

3.3.2 Policy Specification in CCA

As mentioned in the previous section, the ECA model consists of an event to be monitored, condition to be evaluated and an action to be performed. In CCA, the ECA model can be specified in a natural fashion using the context-guarded process, as follows:

$$! \ \alpha(\tilde{x}).(e \wedge c)? \ a$$

Where $\alpha(\tilde{x})$ is an input capability. The parameters e, c and a stand for event, condition and action respectively. The event e can be represented as a context

expression [3]. The condition c can also be represented as a context expression. In the same manner, the action a is also a process. It may represent a single process or a set of processes. A set of processes is represented as: $(a_1 \mid a_2 \mid a_3 \ldots)$. A more complex policy can be represented as a set of policies running in parallel as follows:

$$policy_1 \mid policy_2 \mid policy_3 \mid \ \ldots \ \mid policy_n$$

Example 3.3.1

Let us suppose a system's policy states the following: 'Upon system's infection, execute a system back up process'. This system can be specified as follows:

$$! :: (sys\_state).\ (sys\_state = infected)?\ \texttt{back\_up}.\ \mathbf{0}$$

Where the system receives the systems state at all times and it checks whether the system state equals to infected; the system then executes the process of `back_up`*, which is an abstract process representing the performing of a system's back up.*

An access control policy is used to grant/deny permissions for subjects to perform specific actions on specific objects. In this context, there are three types of authorisation policies [94]:

- *Positive authorisation:* used to explicitly grant permission upon a request made by a subject to perform an action on a specific object. In this policy, permission is granted if explicitly stated; otherwise it is denied. This policy is also called *default negative policy.*
- *Negative authorisation:* used to explicitly deny permission upon a request made by a subject to perform an action on a specific object. In this policy, a

[3] See Table 3.6

permission is denied if explicitly stated, otherwise it is granted. This policy is also called *default positive policy.*

- *Hybrid authorisation:* this type of policy combines both positive and the negative authorisation, in which the default process for such a policy is inactivity, and upon a subject request, a decision is made to either grant or deny that request.

When a policy is used as an access control policy, the action of the policy depends on whether it is a positive or negative authorisation. The action of a positive authorisation is to send a '*grant*' message. By contrast, the action of a negative authorisation is to send a '*deny*' message. Figure 3.2 below depicts a block diagram that shows the components that are associated with an access control system.

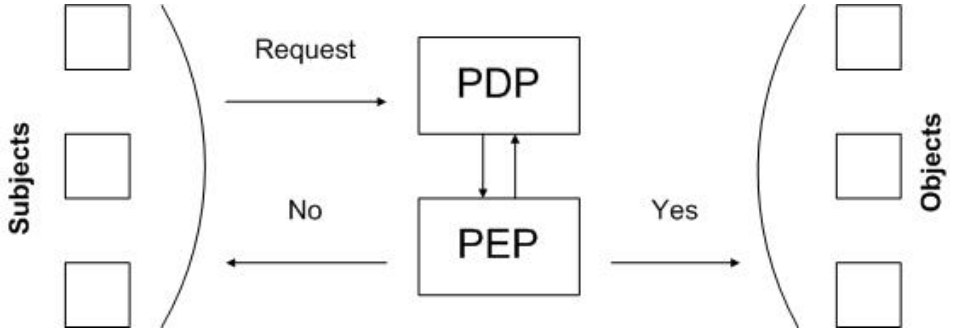


Figure 3.2: Block diagram: access control policies

Example 3.3.2 *A system running a positive authorisation policy consists of objects (subjects request to access), a policy decision point (PDP) and a policy enforcement point (PEP). We demonstrate a simple model for the system as follows, where the specification of the PDP ambient is:*

$$pdp[!s :: (s, o, a).\ \texttt{evaluate}.\ pep :: \langle s, o, a, \texttt{decision}\rangle.\ \mathbf{0}]$$

where s is the subject ID that sends the request, o is the object the subject is trying to access, and a is the action s wishes to perform on o; evaluate is an abstract

process which stands for the the evaluation of whether or not to grant the subject the permission to access the object. The decision is the decision that is made by the PDP. The specification of the PEP component is as follows:

$$pep[!pdp :: (s, o, a, \texttt{decision}). \left\{ \begin{array}{l} (\texttt{decision} = grant)?\ o :: \texttt{action}.\ \mathbf{0} \\ (\texttt{decision} = deny)?\ s :: \langle DENY \rangle.\ \mathbf{0} \end{array} \right\}$$

where it receives the name s of the corresponding subject, the name of the object o, the action a and the decision; then it checks whether the decision is positive (i.e. allow the subject to access the object). If so, the action is performed on object o. However, if the decision is to deny the subject, then the PEP sends the message `deny` *to subject s.*

3.3.3 Context-aware Policy

As mentioned in the previous section, a policy rule in general has an event, a condition and an action. However, a policy in its nature has a static (i.e. fixed) action that is executed basically on a static condition in a dynamic environment (i.e. continuously changing). Many researchers have developed context-based policy models which support policies in pervasive computing applications. Most of these models were in the area of RBAC; in that context, different types of context have been considered as part of building a RBAC system. Contextual information in a RBAC system includes user's presence at a specific location, or with other users, or occurrence of specified environmental conditions. For example, [63] has designed a context-aware RBAC model for pervasive computing application, which was driven by the context-based access control requirements of such an application. In the same context, [100] has introduced a contextual RBAC model, which focuses not only on what the subject wants to do which action on what object, but contributes

by adding the purpose of the action: a context. [32] has done the same work of integrating context into access control policies, by representing context in semantic knowledge (producing context ontology) and extending a standard access control policy language to incorporate the semantic knowledge.

With the advent of context-aware systems, a new definition of a policy has emerged; *context-aware policy* (CAP). Such a policy is able to change its action based on changes in the surrounding environment, and this demonstrates a whole new concept of how policies are executed and in what way. Here, we propose a new model for CAP in general, in which a policy has a condition and an action (i.e. can be expressed using ECA model), and additional *context* parameters. A legitimate action is executed if the condition of the policy is met based on a specific context or history of context. Moreover, we give a proper definition of what a CAP is.

A context-aware policy is a policy that uses previously specified monitored events to trigger the evaluation of a condition in a specific context in order to execute a specific action.

For example, a policy may execute an action, say x, based on the current state of an object, say y, in a specific context κ_1. On the other hand, for the same state of object y but in a different context κ_2, a context-aware policy might execute a different action, say x', considering the change of the context related to that object. A context can represent a location, speed, temperature, resource, other people, etc.

In order to operate a context-aware policy, the system dealing with such policy has to be able to sense the surrounding relevant context. Therefore, when the situation of the surrounding environment (i.e. context) changes, the policies must also reflect this change. To specify such policy, we need to have the suitable notation to

represent it. Fortunately, in CCA we have just the right notion of context κ. Hence, a context-aware policy model can be specified in CCA as follows:

$$! \ \alpha(\tilde{x}).(e \wedge c \wedge \kappa)? \ a$$

Where $\alpha(\tilde{x})$ is an input capability. An event and a context can both be expressed as context expressions, however, their expressions are different. An event, in the perspective of a policy, is some action that has happened within the environment of the system. Events can, but not necessarily, cause state transitions from one state to another. Some examples on an event are: pressing a keyboard button, activating some function, someone enters a room, timer reaches a specific value, music starts to play, etc.

On the other hand, a context is a piece of information that characterises the situation of an entity. Context can be the name, location, resources, or any other piece of information that is relevant to an entity. For example, when an entity changes its name, the action of changing the name is an event, while the new name of that entity is a context. The following example shows how evaluating an event is different from evaluating a context.

Example 3.3.3

Suppose user Sam is in his office, and a policy says that: if Sam changes his location to a conference room, then he should send a 'Hello' message. Therefore, the way to evaluate such policy is by evaluating the event (Sam changes location) and the context (that location is conference room). This policy can be represented in CCA *as:*

$$sam[! \ :: \ \alpha(\tilde{x}). \ (\texttt{location\_changed}(sam) \wedge \texttt{location} = conf\_room)? \ :: \langle Hello \rangle. \ \mathbf{0}]$$

where `location_changed`(sam) *is a context expression which is true when the location of the user sam has changed.*

Example 3.3.4

Assume that the smart mobile phone of the Sam has to switch its profile to the 'silent mode' if user Sam is with friend called Mary. The context that the policy has to evaluate is that user Sam is with friend Mary. It is not a very easy job for the policy to evaluate such a context, because it has to evaluate the person with user Sam; it is not anybody, it is not anybody called Mary, it has to be a friend and called Mary. In `CCA`*, such a policy can be easily evaluated as follows:*

$$phone[\ !\ (\texttt{sam\_with}(mary)\ \wedge\ \texttt{friend}(sam, mary))?\ \texttt{switch\_to}(\texttt{silent}).\ \mathbf{0}]$$

3.4 Summary

This chapter has presented the computational model which is based upon the notion of an ambient. We defined an ambient and illustrated its characteristics which include, but are not limited to, communication and mobility capabilities. Such quality features make an ambient a perfect candidate to represent any entity or object in a context-aware system. Then, we presented the process calculus of `CCA`, which is a mathematical notation suitable for formally specifying an ambient. We illustrated the syntax of `CCA` in terms of processes and capabilities, we explained the semantics, and demonstrated some examples. As mobility and context-awareness are primitive constructs in `CCA`, then `CCA` is the suitable process calculus for modelling context-aware mobile applications.

As part of our contribution, we have demonstrated how to represent policies in `CCA` using the ECA model, and we have proposed a new model for what is called a

context-aware policy and have given it a new proper definition. In the next chapter, we propose a novel policy-based architecture for context-aware systems. Then using `CCA`, we will specify the policy enforcer.

Chapter 4

Policy-based Architecture for Context-aware Systems

Objectives:

- Give an overview of some previous approaches and their limitations
- Propose a novel policy-based architecture for context-aware systems
- State the motivation behind our approach
- Demonstrate two scenarios using our architecture
- Using `CCA`, specify the policy enforcement mechanism in our architecture

4.1 Background

Context-aware systems are systems that have the ability to detect, reason and take action upon contextual information, and provide information and/or services to the users, with respect to their needs [34]. Many architectures and frameworks of context-aware systems have evolved in recent years. Some of the architectures aim to build a secure context-aware system while the rationale behind others is how to deploy an adaptation mechanism to adapt the behaviour of a context-aware system to its environment. However, different uses for the architectures does not overlook the fact that every context-aware system architecture has to have the essential aspects of sensing context, and reasoning upon the context information.

One architecture was presented as part of a project called 'Context-Awareness Sub-Structure' (CASS) [42]. The CASS project aimed at structuring a middleware for context-aware mobile devices. The essential concept behind it is that it senses and interprets the context, storing and retrieving the context information, providing information to mobile devices as well as retrieving information from them. The CASS is basically a middleware built for context-aware mobile devices, for it gets over the difficulty of small processors and storage capacity in mobile devices, however, it is not useful with larger systems. A second issue with the CASS architecture is that it makes use of high-level context information only, because the higher the level of the context, the more easily it can be used. This means that its reasoning dynamism is not able to reason low-level context information, which is a limitation for larger context-aware systems.

Another architecture was proposed as part of the hydrogen project [53]. This architecture was designed especially for mobile devices. The approach of the hydrogen

project is to make a system able to distinguish between local and remote contexts. This architecture is implemented on mobile devices. However, such an approach is limited in terms of speed of processing and storage of contextual information, also it lacks reliability because failure of the device leads to failure of the whole system [1].

Many researchers in other fields have implemented policy-based management or proposed context-awareness to their work. For example, a policy-based management framework has been proposed by [52] for the management of MANETs (Mobile Ad hoc Networks). The framework adopts the policy-based network management (PBNM) proposed by the IETF (Internet Engineering Task Force) [3], along with the concept of context-awareness, to deploy a hierarchical and distributed model for MANET management. Moreover, [75] has proposed a context-aware, policy-based framework for ambient network, it has also adopted the PBNM framework proposed by the IETF organisation. The main purpose behind that framework is to improve the self-adaptability of a system for multimedia services. [25] proposed a generic policy-based self management model that can automatically detect and repair the problems that have appeared during the context adaptation processes.

The list of examples on context-aware system architectures and frameworks is long, however, one recognisable framework to build context-aware systems is the five-layered conceptual framework [8, 37, 7], depicted in Figure 4.1. The layered conceptual framework is a general, initiative step towards building a context-aware system. It has the basic layers required by any context-aware application: sensors, raw data retrieval, preprocessing, storage and management, and the application.

The first layer of the framework consists of a collection of different sensors. These

[1]The figures for the architectures are depicted in Chapter (2)

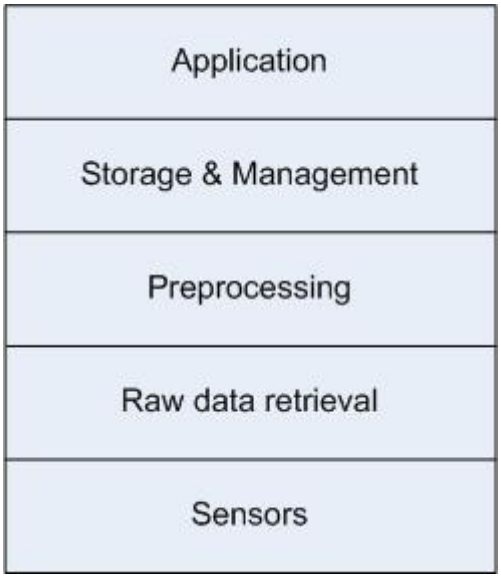


Figure 4.1: Context-aware system framework [8, 7]

are not only a means of sensing hardware (e.g. movement sensor, light sensor, pressure sensor ... etc.). As mentioned earlier in Chapter (2), sensors can be categorised into three different types:

1. Physical sensors
2. Virtual sensors
3. Logical sensors

The second layer is concerned with retrieving raw context data. It makes use of the data collected from the drivers of the physical sensors and the API's of the virtual and logical sensors. The functionality of this layer is often implemented in a way that allows the extraction of exact information needed; this is done by providing more abstract methods (such as `getTemprature()`, `getLocation()`).

The preprocessing layer is not necessarily implemented in every context-aware system, but it may offer useful information for the system if the information from the previous layer (raw data) is not ready to act upon. This layer is responsible

for reasoning and interpreting contextual information; i.e. it raises the information supplied from the previous layer to a new level of abstraction, so that the information can be used to support decisions within the system. For example, a GPS might not be able to give the exact location (floor, room number) for a person, so the system might use other information from other devices to get the exact location of the person. A context-aware system might sometimes combine two or more pieces of contextual information to get the exact information, because combining information might be more accurate. The process of combining information is called *Aggregation* or *Composition*.

The fourth layer, storage and management, is responsible for storing the information, organising the data gathered and offering them to the clients via an interface, via the fifth layer, the application layer.

4.2 Policy-based Architecture for Context-aware Systems

In this section, we propose a novel policy-based architecture for context-aware systems. The term '*Policy-based*' gives the indication that the policies play a major role in constructing the architecture of a context-aware system. Policies are vital to the functionality of context-aware systems, and yet, no approach has been proposed to focus on such a significant issue. For a context-aware system, its policies *may* change, based on the important factor that the environment in which the system is running is continuously changing.

4.2.1 The Architecture

There is an evident need for rules to automate and govern the processes and functions of a system. In general, the goal of a policy-based system is to control the behaviour of the system by employing well-defined rules. Policies are needed to define strictly regulated access to various resources.

Rules are needed in every system to govern its behaviour. So, setting policies for a system to run by reduces the complications of controlling that system. However, in a context-aware environment, the incessant changing of context may affect the policies of a system causing it to change their action. We propose an architecture that provides dynamic control of system actions according to the changes in policies. A context-aware system based on our architectural design operates as follows: it executes an action after evaluating a policy condition against contextual information so that any change in policy rules will not affect the system's design.

There are two important aspects in which our architecture should be policy-based and context-aware. Since our proposed architecture is strictly applied for context-aware systems, we utilise the five layers of the conceptual framework (Figure 4.1) to build the components of our architecture, in which the conceptual framework describes the basic ground of our architecture. The second aspect is the 'policy-based' aspect; generally speaking, for a system to have the characteristic of a policy-based system, it should have four main components [3, 30]: *Policy Management Tool* (PMT), *Policy Repository* (PR), *Policy Decision Point* (PDP) and *Policy Enforcement Point* (PEP). Figure 4.2 below depicts a framework for Policy-Based Management (PBM) proposed by the *Internet Engineering Task Force* (IETF) organisation.

The policy-based management framework is considered as a standard for which a

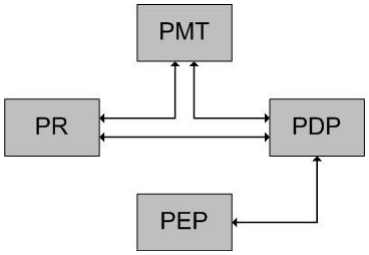


Figure 4.2: IETF's framework for PBM block diagram [30]

policy-based system should have its components and their functionality. Therefore, we propose a policy-based architecture for context-aware systems which encapsulates the conceptual five-layered framework for context-aware systems and the policy-based management framework. Our proposed architecture is depicted in Figure 4.3 below.

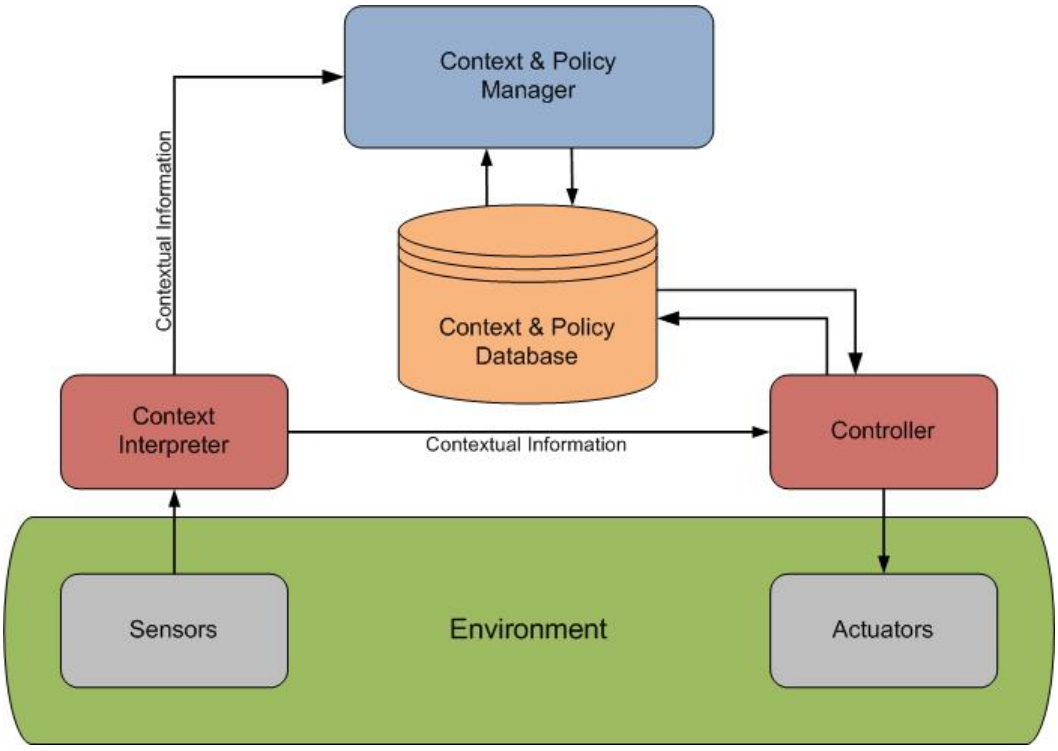


Figure 4.3: Policy-based architecture for context-aware system

The context-awareness potential in a context-aware system falls to three key processes: sensing, reasoning and acting. The sensing is done through the sensors, and the reasoning is the mechanism which takes the decisions to act upon previously

set rules. Our proposed policy-based architecture satisfies those vital processes throughout the processes carried out by its six components; these are:

- *Sensors*
- *Context Interpreter*
- *Context and Policy Manager*
- *Context and Policy Database*
- *Controller*
- *Actuators*

The environment around the system represents the source of contextual information which is the input to the system. The *sensors* and the *actuators* represent the physical layer. The *context interpreter* represents the raw data retrieval layer and, so to speak, a small part of the preprocessing layer. The whole preprocessing layer is signified in the *controller*. Finally, the *context and policy manager*, and the *context and policy database* correspond to the management and storage, respectively. However, the manager, database and the controller components match the policy-based management framework for a policy-based system in terms of components and their functionality. In Figure 4.4 below, we depict an activity diagram which shows the main processes executed by our policy-based architecture.

4.2.2 Sensors

The sensors (physical layer) consist of a number of different sensors integrated into the surrounding environment in which the system is running. The sensors differ in terms of types according to the requirements of the applied context-aware system. This layer concerns the way in which the context data is captured. The sensors

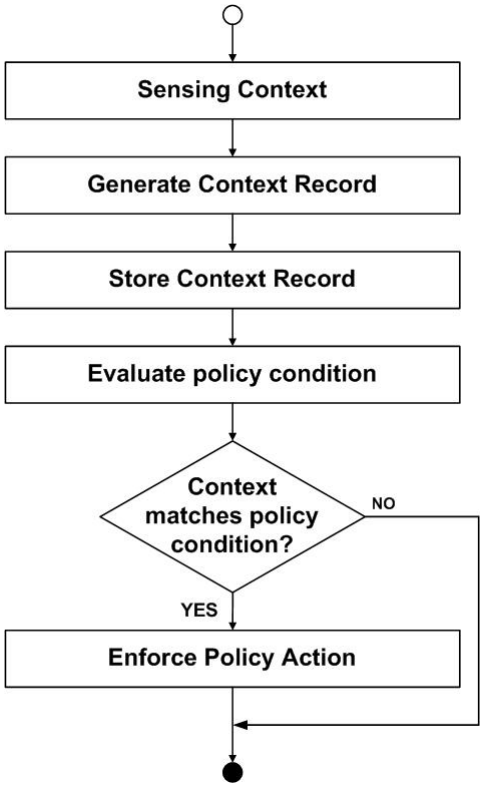


Figure 4.4: Activity diagram

(*physical* and *virtual*) are deployed in this layer to capture the context (e.g. user context, device context, location context, time context, ... etc) and transmit it to the context interpreter. The functionality of the sensors is important to the functionality of the context-aware system, as failure in the sensors leads to faulty information being provided to the system, and thus a malfunction in the system's behaviour.

Figure 4.5 below depicts a detailed view on the sensors layer. Physical sensors (e.g. movement, pressure, light ... etc) and virtual sensors (software application/services, log in/out processes) are used in a context-aware system; however, logical sensors are also important in a context-aware system, but they are not really an actual type of sensors as much as they are *reasoning* processes that combines information from one or more sources in order to solve higher tasks. The arrows from the sensors upwards represent a continuous flow of context.

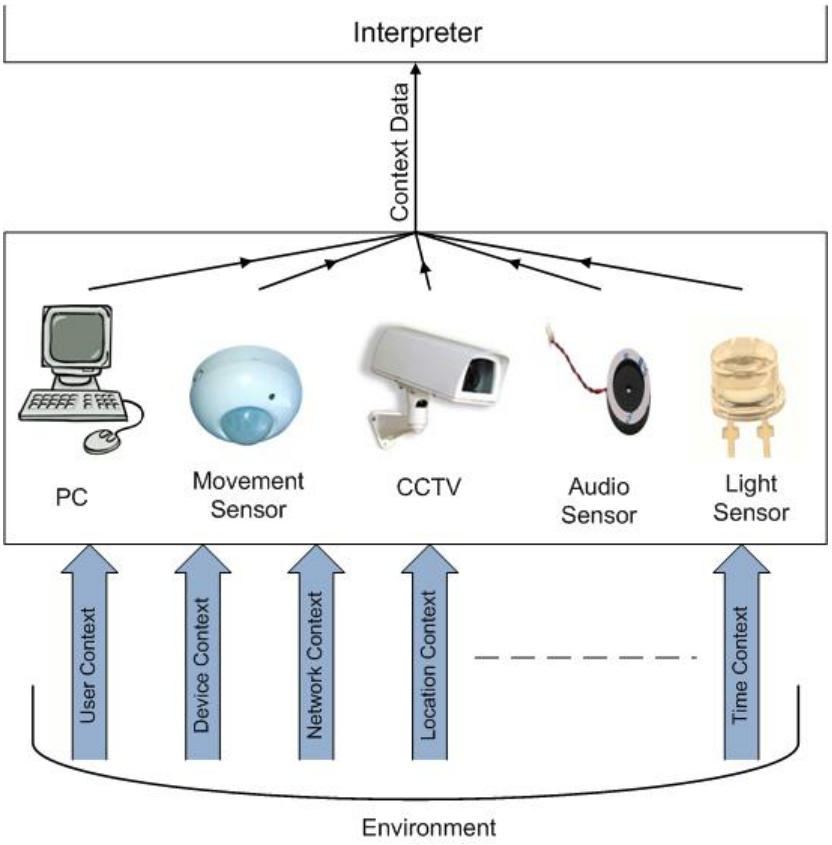


Figure 4.5: The sensors of the policy-based system architecture

The context captured can be separated into two contexts: physical context and logical. The physical context represents the level of physical sensors integrated into the environment, the properties of which are at a very low level of abstraction; thus, the context must be continuously updated taking into account the constantly changing environment states. The logical context represents the more abstract information about the environment (i.e. virtual sensors). Logical context information is needed whenever a change happens in the environment (e.g. GPS coordinates). The sensors captures the context and transmits it to the *context interpreter* for processing.

4.2.3 Context Interpreter

The context interpreter (raw data retrieval layer and preprocessing layer) is the component responsible for generating context records using the context data obtained from the sensors. The generated context records are in the form of:

$$(parameter_1, parameter_2, parameter_3, \ldots parameter_n)$$

where the entity being monitored could be any object (e.g. user, device, weather, vehicle), and $parameter_i$ (where $1 \geq i \leq n$) represents the parameter related to the monitored object (e.g. name, location, nearby resources, value, weight ... etc). For example, for a system that monitors the students and their location within a university, a generated context record depending on the system's requirements may be in the form of $(studentID, location, timeStamp)$.

The process of generating context records undergoes two phases, as shown in Figure 4.6 below, checking the context followed by generating context records. Two components are responsible for these two processes; *interpreter engine* and *records generator*.

The interpreter engine receives the context and immediately checks to see if the context is sufficient or not, where *insufficiency* for context data means that the context is useless on its own (e.g. the ID of a user is useless unless convoy with location, age, etc.). If the context is sufficient, the interpreter engine passes the context on to the records generator. If, however, the context is not sufficient, the interpreter engine performs the process of *combination / aggregation*, which entails combining two or more context data to produce useful, meaningful context information.

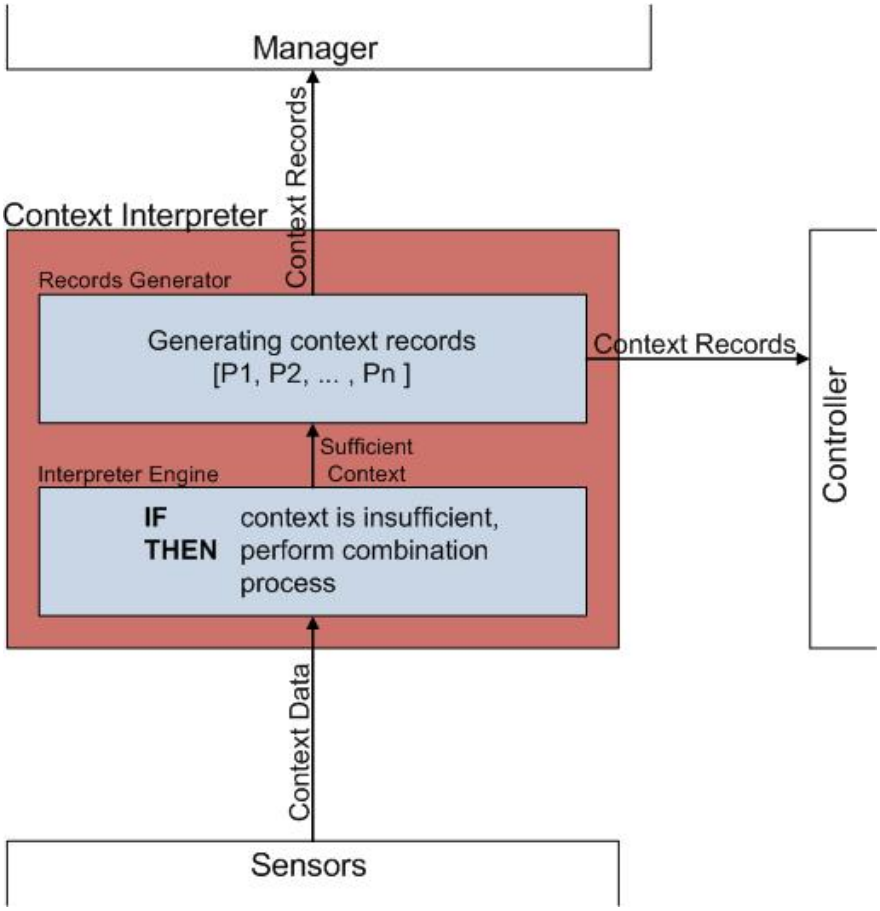


Figure 4.6: The interpreter of the policy-based system architecture

For example, a virtual sensor may detect a login process for a user on some computer, but for that piece of context information the system will not be able to locate that user (assuming there is no CCTV), so by *combining* the computer ID number the user is using, along with the device database mapping, the system can identify the location of the computer, and thus the location of the user. After continuously generating the context records, the context interpreter sends them to the context manager and the controller so that they can be stored, processed and evaluated.

4.2.4 Context and Policy Manager

The context and policy manager (management layer) is the management unit of the policy-based context-aware system. It has two components: *a context manager* and *a policy manager*.

The context manager is responsible for categorising, editing, storing and administering context information. It receives the context records generated by the context interpreter, and then edit what needs to be edited, categorises the context, and stores the context into the context database. The processes of the context manager are controlled by authorised administrator(s).

The policy manager is the interface from which an administrator can edit, store and manage policies. The policies of the system can be inserted and updated manually through the policy manager. It can store/retrieve policies into/from the policy repository (database). The simplest way to specify and form a policy in general is to use the traditional formula of `Event, Condition, Action` (ECA), where on *event*, if *condition* is satisfied then *action* is performed. A policy rule is an essential component of a policy-based system; it represents a set of actions in regards to a set of conditions where the conditions are triggered and evaluated to determine whether the actions should/should not be executed. Figure 4.7 below depicts a close view on the context and policy manager of architecture and associated connections.

4.2.5 Context and Policy Database

The context and policy database (storage layer) is an integral part of a policy-based context-aware system. It encapsulates context information and policy information. The database is a data store in which the manager can store context information

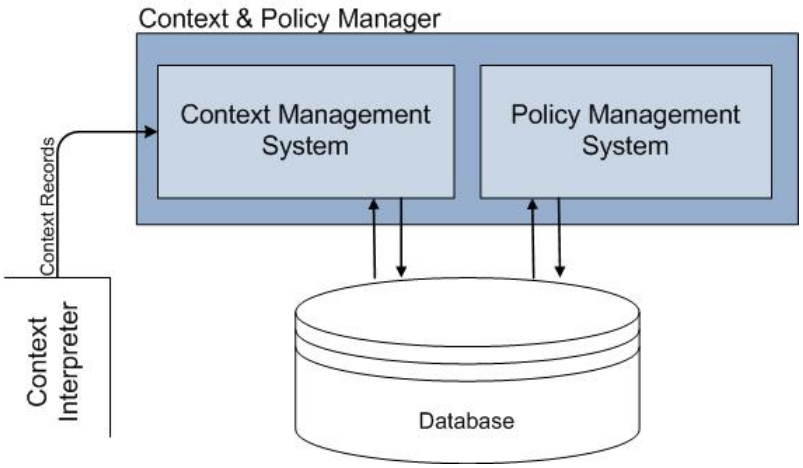


Figure 4.7: The manager of the policy-based system architecture

and policy information, which can be retrieved by the manager itself or the PDP for evaluation. Only the manager has the authorisation to have access to maintain the database (store, edit, or retrieve), because a single point of failure of policy makes a policy-based system vulnerable. E.g. a failure in access control policy may cause unauthorised access to confidential information which would compromise the system.

Figure 4.8 allows close examination of the database connections with other components of the policy-based system.

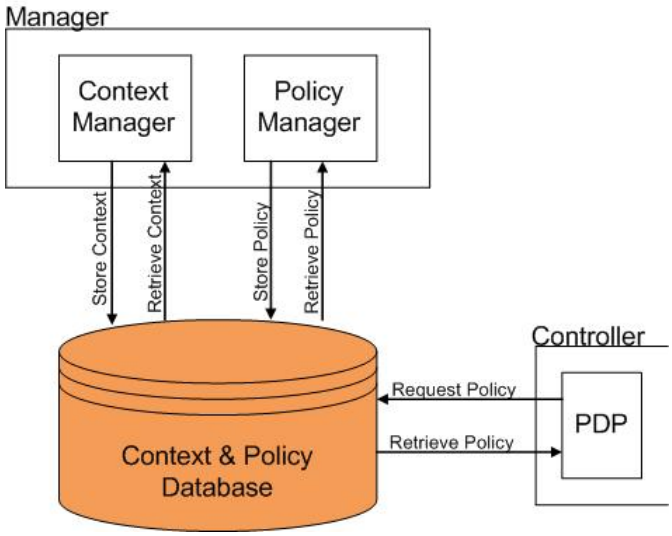


Figure 4.8: The database of the policy-based system architecture

4.2.6 Controller

The controller (management layer and application layer) is an essential component which gives the architecture of a context-aware system the policy-based functionality. It is responsible for making decisions and enforcing them, based on the context records obtained from the context interpreter and the policies obtained from the database, respectively. The controller is the engine in which the decisions take place and the policy actions are enforced. The controller has three connections, *(i)* it receives context records from the context interpreter, *(ii)* it has access to the context and policy database and *(iii)* it sends the decision results to the actuators.

The controller consists of two components: *Policy Decision Point* (PDP) and *Policy Enforcement Point* (PEP). The procedure of making a policy decision takes into account two aspects; one deals with the evaluation of a policy rule's condition, the other with the enforcement of a policy action when condition for a policy matches a context. Figure 4.9 below shows a detailed image of the controller and its internal components and connections with other components of the system.

The PDP is a logical entity responsible for evaluating policy condition. It receives context records from the context interpreter, and it has full access to the context and policy database in order to evaluate the current context in accordance with the relevant policy condition. The ability to access fully the database gives the impression that the policies exist in the PDP as well. The PDP is responsible for making policy decision and passing these decisions to the PEP.

The PEP, in its turn, is responsible for enforcing policy actions based on the policy decisions obtained from the PDP. The policy enforcement process is done by imposing the actions through the actuators (physical layer) which are integrated

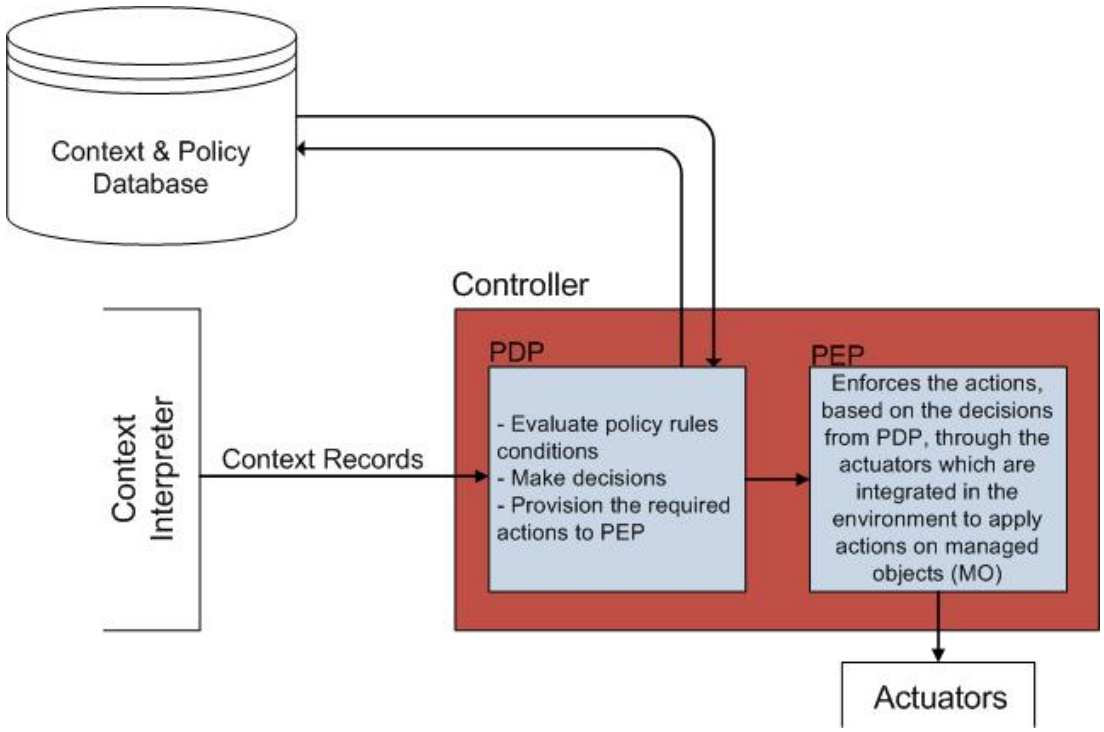


Figure 4.9: The controller of the policy-based system architecture

within the environment.

4.2.7 Actuators

The actuators (physical layer) are part of the physical layer, they consist of physical entities integrated into the environment (objects). The actuators are integrated into the objects to control them. They are responsible for receiving policy enforcement results from the controller (PEP), and applying the actions on the Managed Objects (MO).

4.3 Scenarios

In this section, we present two simple scenarios that show the relevant aspects of our approach functionality. Sequence diagrams are used to explain each scenario.

The sequence diagrams are represented in a visionary fashion which demonstrate the transition steps from one functional component of the system to the next.

Scenario 1: Let us suppose an air conditioning system is used to keep a fixed temperature inside a room. It operates as follows; it monitors the room's temperature via a specific sensor installed in that room. However, if the temperature of the room is above the degree permitted, then the system turns down the thermostat, on the contrary, if the room's temperature is under the desired degree, then the system turns the thermostat up. So, the system simply senses the temperature in the room and eventually adjusts the thermostat to maintain the temperature degree required. Figure 4.10 shows a sequence diagram demonstrating how the temperature maintaining system functions.

Scenario 2: Suppose a smart mobile phone that should turn itself to the 'silent mode' when the user carrying the phone walks into a conference room. So, the system works as follows: it constantly monitors the location context of the user carrying the phone. When the location context eventually changes to match a conference room, this will trigger the policy action, which is to turn the smart mobile phone to the silent profile. Figure 4.11 shows a sequence diagram of how the system will react if the location context is changed to a conference room.

4.4 Policy Enforcement

In this section, we give an abstract specification for the policy enforcer used in our proposed policy-based architecture. A policy enforcement mechanism is the method of enforcing a policy specific action. Our policy enforcement mechanism consists of a PDP, PEP and a database as shown in Figure 4.12.

The above diagram depicts the *controller* part of our proposed policy-based ar-

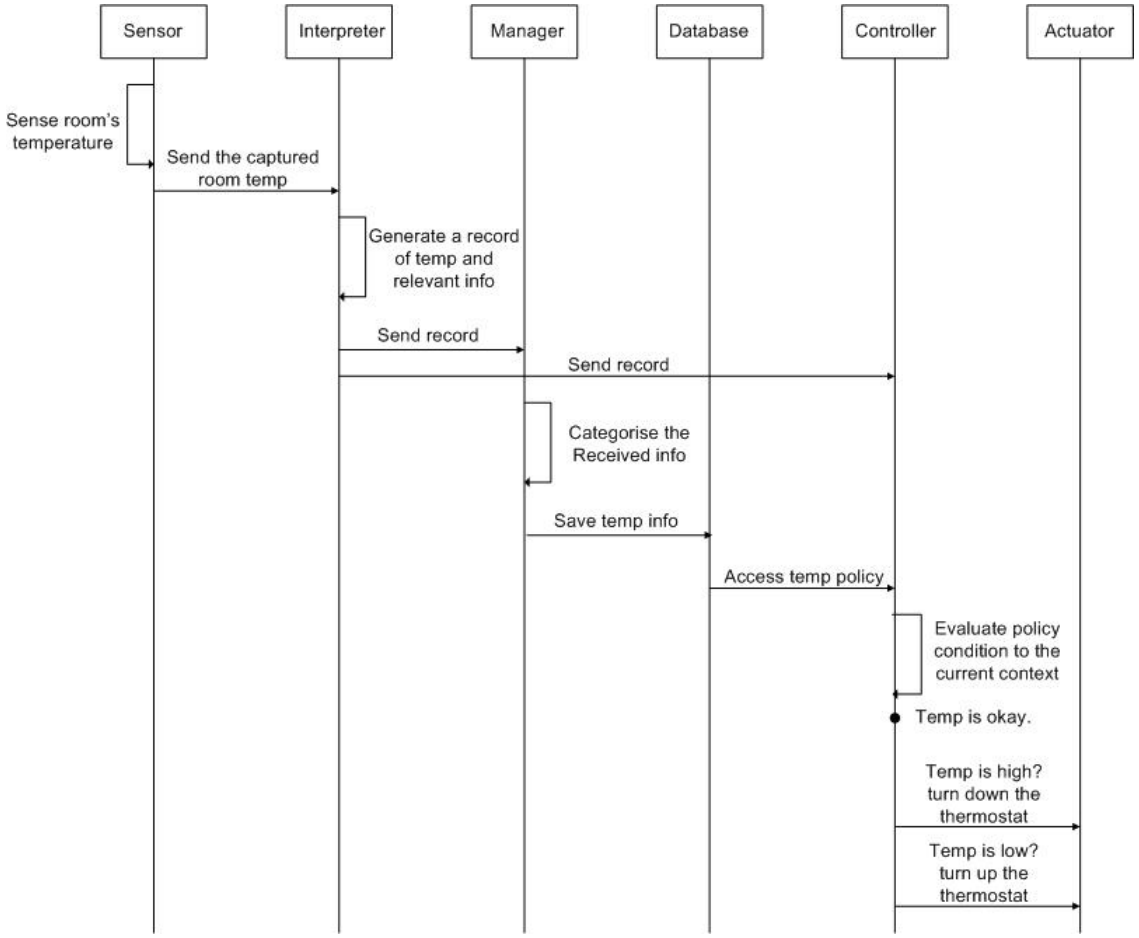
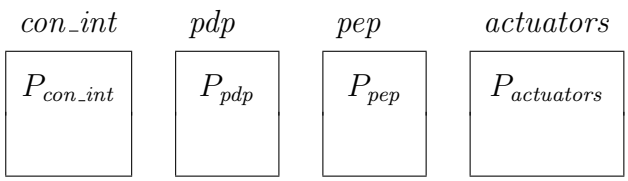


Figure 4.10: Sequence diagram: scenario 1

chitecture. Using CCA notation, we will specify our policy enforcer. First, we assume that the PDP and the database are acting as one entity (in our architecture, the PDP has full access to the policy database.), and the *context interpreter*, in any other system, could be an entity that provides the current context to the PDP. A CCA graphical model of the policy enforcer is depicted in the following figure:



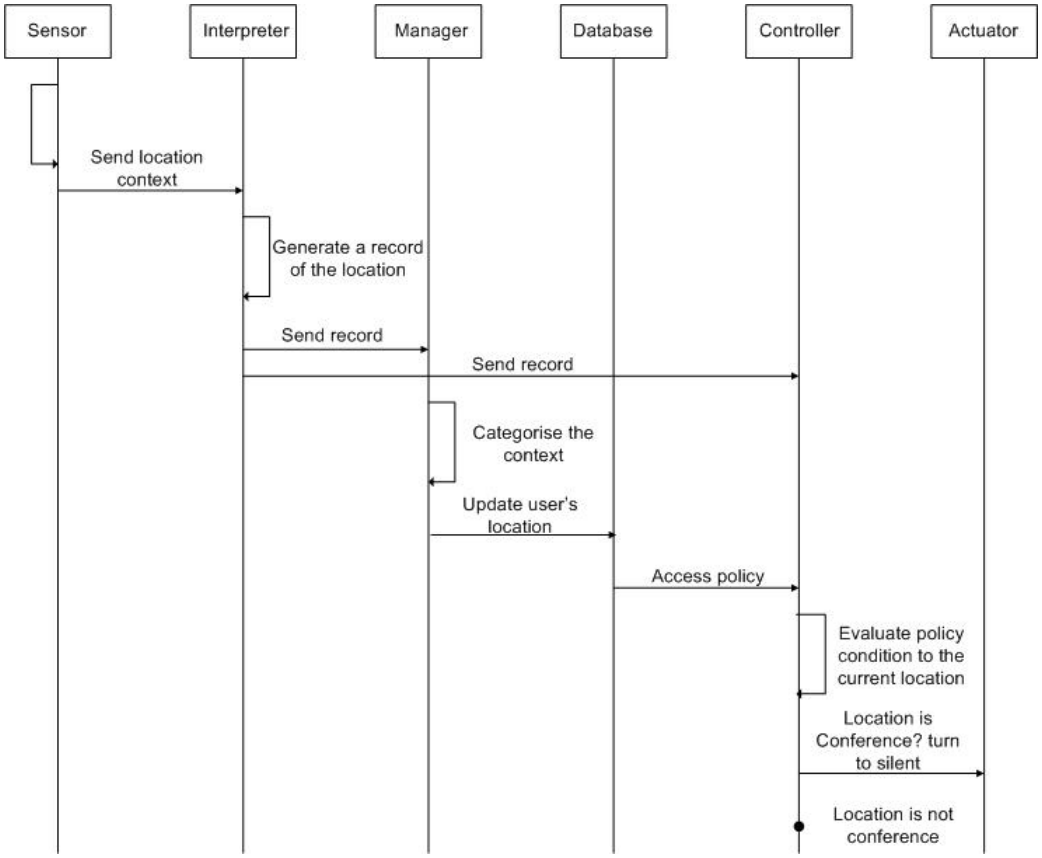


Figure 4.11: Sequence diagram: Scenario 2

The context interpreter *con_int* is modelled in `CCA` as follows:

$$cont\_int[! :: (x_1, x_2, \ldots, x_n).\ P.\ pdp :: \langle y_1, y_2, \ldots, y_m \rangle.\ \mathbf{0}]$$

Here the context interpreter receives from some ambient a number of context information $(x_1, x_2, \ldots, x_n)$. P is the process of generating context records; then it sends the generated context record (assuming that $y_1, y_2, \ldots, y_m$ represents a context record) to the *pdp* ambient.

Thus, the *pdp* ambient by its turn receives a context record from the *con_int* ambient, and evaluates the context with the corresponding policy condition(s). If the current context triggers a policy condition, represented in this case by the process

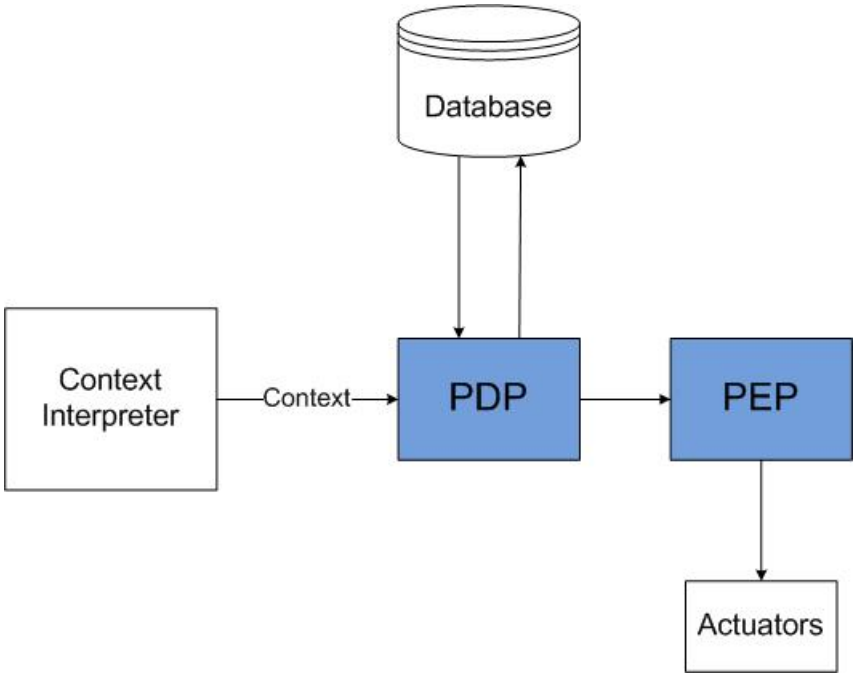


Figure 4.12: Block diagram: our policy enforcer

$\texttt{match}(y_1, y_2, \ldots, y_m, a)$ which is a process of matching the current context with the policy condition and a is the corresponding action to follow, the *pdp* then sends a message containing the required action to the *pep* ambient then it terminates. The *pdp* is modelled as follows:

$$pdp[!\ con\_int :: (y_1, y_2, \ldots, y_m).\ Q.\ (\texttt{match}(y_1, y_2, \ldots, y_m, a))?\ pep :: \langle a \rangle.\ \mathbf{0}]$$

The *pep* ambient receives the required action from the *pdp* ambient, does some calculations (R), and then makes a *process call* to the corresponding actuator ambient. It then terminates, as follows:

$$pep[!\ pdp :: (action).\ R.\ actuator :: \texttt{perform}(action).\ \mathbf{0}]$$

4.5 Summary

In this chapter we proposed a novel policy-based architecture for context-aware systems. Our architecture is based on the five-layered conceptual framework of context-aware systems. Among the architecture, we integrated a policy-based management standard to reason and enforce policy decisions. The motivation behind our architecture is simply to maintain the architectural design of a context-aware system when the policies of the system change. Finally, we specified a simple model of the policy enforcer of our architecture using the mathematical notation of `CCA`, which was presented in the previous chapter (Chapter 3). The next chapter (Chapter 5) will present the formalisation of a real-world case study of an infostation-based mLearning system which is presented in Appendix B.

Chapter 5

Formalisation

Objectives:

- Presenting the infostation-based mLearning system briefly
- Presenting a graphical model of the infostation-based mLearning system
- Using `CCA`, presenting the formalisation of:
  - AAA service
  - mLecture service
  - mTest service
  - IMN service
- Mapping the mLearning model onto our policy-based architecture

5.1 The Infostation-based mLearning System

The architecture of the infostation-based mLearning system is depicted in Figures 5.1. The users within the infostation-based network can access a number of mServices through their mobile devices, via a set of infostations deployed around a university campus.

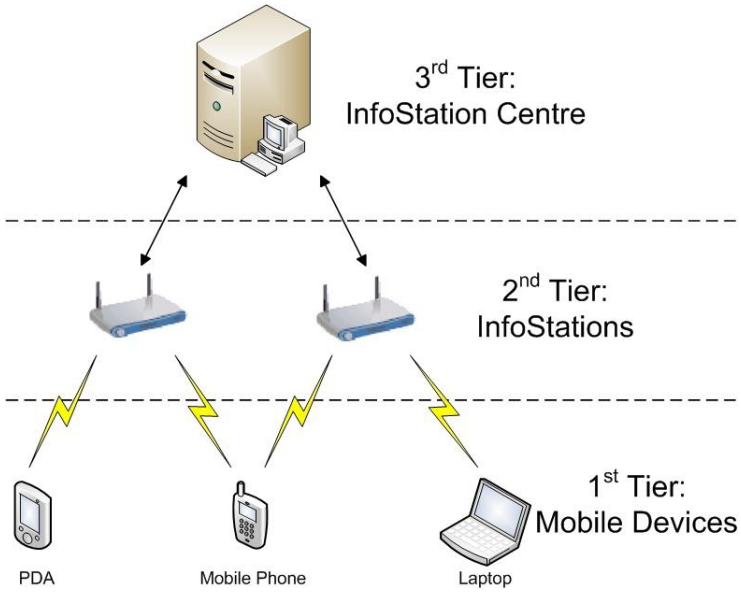


Figure 5.1: The three tiers architecture of the infostation-based network [45]

The three tier network architecture consists of three essential entities: mobile devices, infostations and an infostation centre. The users (carrying mobile devices) must first register with any infostation among the system, and if successful, they can then request a service among the available mServices. The processes of registering and requesting a service is done via the users' mobile devices. Each mService requested is delivered to the user's mobile device *eventually* in the most appropriate format to suit the user's device type (i.e. capabilities of the mobile device), regardless of their location within the system.

The following gives brief information about the mobile services [45, 48]:

- **mLecture:** This service provides the users with the lectures material which they can request and access via their mobile devices.
- **mTutorial:** This service is utilised to increase the students' knowledge in a specific subject; it helps the users to take some form of self-assessment test and get the feedback for it, hence the name. This service combines both the mTest and the mLecture services.
- **mTest:** A test is crucial process to a learning process in general, so as to this mLearning system. It provides a means of evaluating the students' knowledge and giving a valuable feedback. The mTest allows the users to take their exams and submit the results via their mobile devices.
- **Private chat:** The idea of this service is to enable two (or more) users within the same infostation range to chat privately to each other. This is a compositional service, as it is deployed on two tiers: mobile devices and an infostation.
- **Intelligent message notification:** This service enables a user to send SMS, MMS, e-mails, etc., to another user within the range of any infostation within the system.
- **Intelligent phone calls:** This service allows users to make phone calls to each other within the range of the university campus, using *VoIP* technology.

For detailed services and architecture of the mLearning system, see Appendix B.

5.2 Modelling The InfoStation-based mLearning System in CCA

As depicted earlier in Figure 5.1, the infostation-based mLearning system's architecture consists mainly of an infostation centre (ISC), multiple infostations (ISs) and multiple user devices (Hand-set phones, PDAs and laptops). However, we will propose a model for the mLearning system, where each component of the system is modelled as an ambient. That is, the ISC, the ISs and each user device is modelled as an ambient. In this section, we will depict our ambient-based model of the infostation-based mLearning system, and show how this graphical model can be converted into CCA textual representation.

To model the infostation-based mLearning system, we represent each component of the system (i.e. ISC, ISs and user devices) as ambient. For the sake of abstraction, we will model the user device and the user holding it as one entity (i.e. one ambient). When a user makes a request, the PA installed on the user device sends the request to the IS on behalf of the user, but in our model, the user and device act as one entity. For example, a device, say PC, being used by a user, say 303, is modelled by an ambient named $PC303$. The ISC ambient is a central component of the system and runs in parallel with the IS ambients, and the user devices within the range of the an IS are child ambients of that IS ambient as illustrated in Figure (5.2).

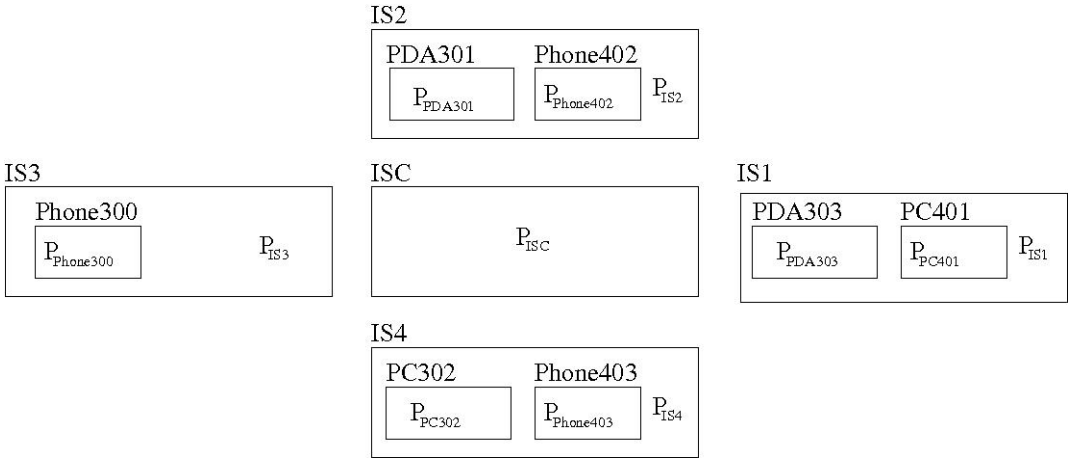


Figure 5.2: A model of the infostation-based mLearning system

The depiction of a user device inside an IS means that the user is inside the range of this IS. Which means, a user can have the privilege of using the services of the system as long as he/she is inside the range of one of the ISs. Moreover, a user can move freely among the ISs of the system (i.e. mobile). The above figure is represented textually by the following `CCA` process:

$$
\begin{aligned}
ISC[P_{ISC}] \quad &| \quad IS1[PDA303[P_{PDA303}] \mid PC401[P_{PC401}] \mid P_{IS1}] \\
&| \quad IS2[PDA301[P_{PDA301}] \mid Phone402[P_{Phone402}] \mid P_{IS2}] \\
&| \quad IS3[Phone300[P_{Phone300}] \mid P_{IS3}] \\
&| \quad IS4[Phone403[P_{Phone403}] \mid PC302[P_{PC302}] \mid P_{IS4}]
\end{aligned}
$$

Where each P_x is a process modelling the behaviour of the corresponding ambient x.

For now, our depiction of the mLearning system is quite done in a high-level abstraction form, through which we can have an idea of how to model the system generally, but can not show how the internal components of the ISC and the ISs interact between each other. However, the next section (5.3) specifies in more details what are the components associated with the ISC and the ISs.

5.3 Formalising the mLearning System

Here, we illustrate the formalisation process of the infostation-based mLearning system. The formalisation will be done to the policies of the mServices (AAA, *mLecture*, *mTest* and *IMN*)services offered by the mLearning system. We will first give the model of the ISs and formalise it based on how its components interact when receiving a user request. Then we will depict the model for the ISC and formalise it as well. The formalisation process will be fully done using the mathematical notation of `CCA`.

5.3.1 Notation

The following naming conventions are used in the formalisation process of the infostation-based mLearning system, to differentiate between the variables' names and the constants' names. A variable name begins with a lowercase letter, while a constant name begins with a number or or an uppercase letter. The list of constants names are given in Table 5.1. And the list of the variables' names are given in Table 5.2.

5.3.2 Infostation

An abstract model of an infostation IS_i (for some integer i) is shown in Figure 5.3, where the main components of any IS throughout the mLearning system are the AAA request ambient $AAAreq_i$, the lecture ambient $Lectreq_i$, the test ambient $Testreq_i$, the IMN request ambient $IMNreq_i$ and the cache ambient $Cache_i$. We have not modelled the users' devices as permanent ambients inside the IS for the users will be in and out, and moving around across the ISs of the system, eventually.

There are many ways in which we can model the inside ambients of the infostation-

Table 5.1: Constants

Constants	
Notations	**Description**
ISC	the InfoStation Centre
IS_i	the InfoStation i; e.g. IS_1
$Phone$	a hand-set phone
PDA	a PDA
PC	a PC
$SLIST$	list of mServices
ACK	an acknowledgment
NULL	empty message
DENIED	request denied

based mLearning system. We have chose to model four permanent ambients in each infostation ($AAAreq_i$, $Lectreq_i$, $Testreq_i$ and $IMNreq_i$). Each of those ambient is responsible for dealing with requests made by users' devices for the corresponding service. The advantage of modelling an ambient for every service is that, the IS can directly process the requests without additional computations.

The infostation is a parent to the inside ambients, which are siblings to each other. When a user device comes inside the range of an infostation, it will also be a child ambient to the IS and a sibling to the inside ambients. The $Cache_i$ ambient contains temporarily copies of the lectures and tests previously requested by users for future simple and quick access. The lectures are represented by an ambient named after each lectures's ID (e.g. $Lect001$, $Lect002$), and each of which contains three ambients named $Phone$, PDA and PC, where each holds a convenient format of the lecture for the corresponding type of device. The tests ambient is structured in the same way of the lectures ambients (noting the difference in names). The specification of each of these ambients is given below.

Table 5.2: Variables

Variables		
Notations	**Description**	**Values**
uid	a user's ID	301, 302, 303
suid	a sender's ID	301, 302, 303
ruid	a recipient's ID	301, 302, 303
dtype	a user's device type	Phone, PDA, PC
sdtype	a sender's device type	Phone, PDA, PC
rdtype	a recipient's device type	Phone, PDA, PC
aname	an ambient's name	Phone301, PC303
saname	an ambient's name (sender)	Phone301, PC303
raname	an ambient's name (recipient)	Phone301, PC303
lectid	a lecture's ID	Lect001, Lect002
testid	a test's ID	Test101, Test102
serviceid	*lectid* or *testid*	Lect001, Test102
service	*Lectures* or *Tests*	
reply	a reply to a request	OK, DENIED, content
content	lecture's/test's content	CONTENT
slist	list of services	SLIST
ack	acknowledgment	ACK

5.3.2.1 $AAAreq_i$ Ambient

As mentioned before, each user has to register with an IS in order to have the privilege of accessing the mServices offered by the system. The process of registering is called the AAA[1]. We modelled an ambient inside each IS called the $AAAreq_i$ which is responsible for handling AAA requests sent by user devices willing to register with the InfoStation IS_i. To start the process of registering a user's device, the $AAAreq_i$ ambient has to receive a request from a user's device, which then it forwards it to the InfoStation immediately, this request holds three parameters which are: *uid*, *dtype* and *aname*. The user ID *uid* is for the ISC to create a profile to that user

[1]Authorisation, Authentication and Accountability

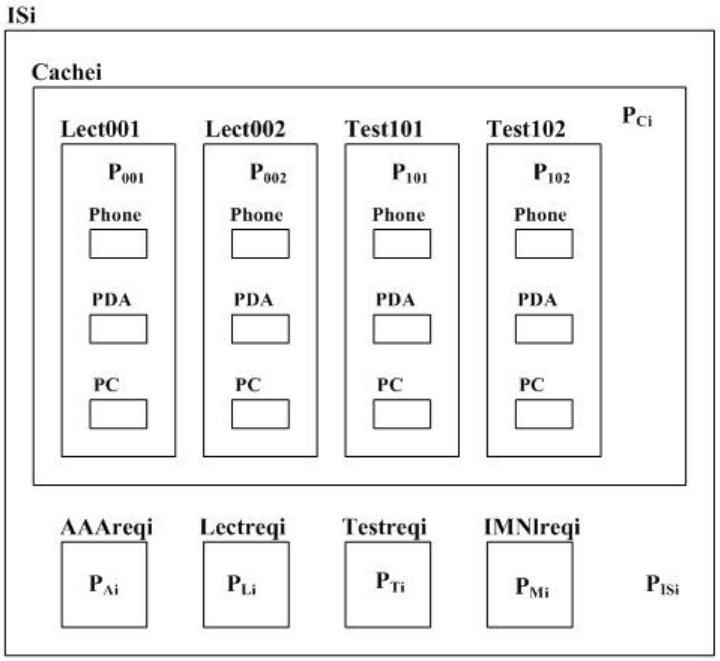


Figure 5.3: A model of an Infostation

in the name of his/her *uid*, the device type *dtype* is to be used by the IS to fetch the corresponding format of the list of services that suits the user device, finally, the *aname* is for the IS to recognise where to send the reply information (i.e. to which ambient). Then the $AAAreq_i$ receives a reply from the InfoStation and again, forwards it to the user's device. This time the reply received from the IS has the form of: (*ack*, *aname*, *slist*). The *ack* is to acknowledge the user that he/she has been registered with the corresponding IS, the *aname* is for the $AAAreq_i$ ambient to know which user ambient to send the reply to, and the *slist* is the corresponding list of services to be sent to the user device. This behaviour is modelled by the following process:

$$P_{A_i} \;\widehat{=}\; !::(uid, dtype, aname).IS_i \uparrow \langle uid, dtype, aname \rangle .$$
$$IS_i \uparrow (ack, aname, slist).aname :: \langle ack, slist \rangle .\mathbf{0}$$

where *uid* is the user ID, *dtype* is the device type and *aname* is the name of the

ambient sending the request.

We modelled that once a user device sends a request to register with an IS, it can not go out of that IS unless it receives a reply from that IS. This is done by not putting the process of sending a request and the process of receiving the reply in parallel, on the contrary, we modelled them as sequential processes.

After a user device is registered with an IS, the user can then request any of the services sent to him/her among the list of services.

5.3.2.2 $Lectreq_i$ Ambient

This ambient handles all the mLecture service requests sent by the user devices. It receives a lecture request from a user device and forwards it to the infostation IS_i. A request for a lecture should contain the following parameters: (*lectid*, *uid*, *dtype* and *aname*). The *Lectid* parameter represents the lecture the user requested. The *uid* parameter is used by the ISC to check if the user is using the mTest service or not. The other two parameters are for the same reason as for the $AAAreq_i$ ambient. This context record generated by the user device (PA) has to contain those parameters and in that specific order or it will result in an error. This behaviour is modelled as follows:

$$! :: (lectid, uid, dtype, aname).\ IS_i \uparrow \langle lectid, uid, dtype, aname \rangle.\mathbf{0} \tag{5.1}$$

Then it gets the reply from that infostation and forwards it to the user device which initiated the request. The reply received from the IS contains three parameters which are: (*lectid*, *reply* and *aname*). The *lectid* parameter is to be sent to the

user because the user could have requested more than a lecture at the same time. The *reply* parameter contains either the content of the requested lecture or a denial response.

$$!IS_i \uparrow (lectid, reply, aname).aname :: \langle lectid, reply \rangle.\mathbf{0} \tag{5.2}$$

So the whole behaviour of the $Lectreq_i$ ambient is:

$$P_{L_i} \mathrel{\widehat{=}} \text{Eq. (5.1)} \mid \text{Eq. (5.2)}$$

5.3.2.3 $Testreq_i$ Ambient

This ambient is responsible for handling all the test requests made by user devices. It receives a test request from a user device and forwards it to the infostation IS_i. Like the $Lectreq_i$ ambient, a request for a test should contain the following parameters: (*testid*, *uid*, *dtype* and *aname*). The *testid* parameter represents the test the user requested. The other three parameters have the same usage as the latter three parameters of the lecture request. This behaviour is modelled as follows:

$$! :: (testid, uid, dtype, aname).IS_i \uparrow \langle testid, uid, dtype, aname \rangle.\mathbf{0} \tag{5.3}$$

It, then, gets a reply from the infostation and forwards it to the user device which initiated the request. The reply received by the user device contains the parameters *testid* and *reply*, the latter parameter contains the test material or a denial response. This can be modelled as the following:

$$!IS_i \uparrow (testid, reply, aname).aname :: \langle testid, reply \rangle.\mathbf{0} \tag{5.4}$$

So the whole behaviour of the $Testreq_i$ ambient is

$$P_{T_i} \mathrel{\widehat{=}} \text{Eq. (5.3)} \mid \text{Eq. (5.4)}$$

5.3.2.4 $IMNreq_i$ Ambient

The $IMNreq_i$ is the ambient responsible for handling the IMN (Intelligent Message Notification) requests made by user devices. For the sake of simplicity, we assume that all the messages send via this service are *text* messages, therefore, every type of device defined in our system can send and receive them. This ambient receives a request from a user device and forwards it to the infostation IS_i. The request for an intelligent message notification is different from other notifications in terms of the parameters associated with the request. A request contains the parameters: (msg, rid, $raname$, sid, $saname$). The msg is the message the user (sender) wishes to send. The rid is the recipient ID which is used then by the ISC to check if the recipient is using the mTest service or not. The $raname$ is the ambient name of the recipient to send the message to. The sid is the sender ID to be used by the ISC to check whether the user is using the mTest or not. The $saname$ is the ambient name of the sender which is used to send the reply to the sender.

$$! :: (msg, rid, raname, sid, saname).IS_i \uparrow \langle msg, rid, raname, sid, saname \rangle.\mathbf{0} \quad (5.5)$$

The $IMNreq_i$ ambient may receive a request from an IS to deliver a message to a recipient or to deliver a reply to a sender, this process is modelled as follows, where the parameters (x, y, z) are the alternatives of the values $(msg, sid, raname)$ or $(reply, rid, saname)$ respectively. $(msg, sid, raname)$ are the parameters used to deliver a message to a recipient, and the parameters $(reply, rid, saname)$ are for delivering a reply to a sender, where the $reply$ could be an acknowledgment of a

message delivery or a denial response.

$$!IS_i \uparrow (x, y, z).z :: \langle x, y \rangle.\mathbf{0} \tag{5.6}$$

Then it receives an acknowledgment from a user device (recipient) and forwards it to the infostation. An acknowledgment message contains a *reply* which is an acknowledgment of delivery, and the *sid* to check if the sender (to deliver the acknowledgment to) is using the mTest or not. The IS will then add the parameter *raname* for the sake of the sender knowing the source of that acknowledgment.

$$!raname :: (reply, sid).IS_i \uparrow \langle reply, sid, raname \rangle.\mathbf{0} \tag{5.7}$$

The whole behaviour of the $IMNreq_i$ ambient is:

$$P_{M_i} \mathrel{\widehat{=}} \text{Eq. (5.5)} \mid \text{Eq. (5.6)} \mid \text{Eq. (5.7)} \tag{5.8}$$

5.3.2.5 $Cache_i$ Ambient

The $Cache_i$ ambient is a database that stores some documents temporarily. It is the ambient where the InfoStation stores copies of requested lectures and tests for future rapid access. The lectures and tests ambients are modelled exactly in the same way. They model a cache memory. A lecture / test is stored as an ambient (named after that lecture's id / test's id) which contains three persistent memory cells (see Example 3.1), each containing a version of the lecture / test suitable to a specific type of device (phone, PDA or PC). When an InfoStation receives a mLecture / mTest service request from a user device, it checks for the requested material in its cache first rather than getting it from the InfoStation Centre directly. The process of checking the availability of a lecture inside the cache is done by sending a request to the $Cache_i$ ambient which then checks whether it has the ambient of the requested

lecture or not. If the requested lecture is available the cache ambient retrieves it and sends it back to the InfoStation, otherwise, it replies immediately to the InfoStation that this lecture does not exist. The behaviour of the $Cache_i$ ambient is modelled by the following process, where the parameter *serviceid* means either *lectid* or *testid*:

$$P_{c_i} \;\widehat{=}\; !\uparrow(serviceid, uid, dtype, aname).$$
$$\left\{\begin{array}{l} \texttt{has}(serviceid)?serviceid{\downarrow}\langle dtype, aname\rangle.\ serviceid{\downarrow}(reply, aname). \\ \uparrow\langle serviceid, uid, dtype, reply, aname\rangle.\mathbf{0} \mid \\ \neg\texttt{has}(serviceid)? \uparrow\langle serviceid, uid, dtype, NULL, aname\rangle. \\ \uparrow(serviceid, content, dtype).\ serviceid{\downarrow}\langle content, dtype\rangle. \\ serviceid{\downarrow}(ack). \uparrow\langle ACK\rangle.\mathbf{0} \end{array}\right\} \tag{5.9}$$

The lecture / test ambient (named after the lecture's id *lectis* or test's id *testid*) in the cache acts as follows, where *serviceid* means either *lectid* or *testid*, the *serviceid* ambient receives a request from the $Cache_i$ ambient containing the device type and the ambient name of the user. The *serviceid* ambient then checks if it has the corresponding format of the requested lecture / test and reply accordingly. If it has it, then it forwards the contents to the $Cache_i$ ambient. If not, it replies with a $NULL$ message and waits for the IS to fetch the corresponding format of the lecture / test and forwards a copy to *serviceid* ambient so it can store it for future access. This behaviour is modelled as the following:

$$P_{serviceid} \;\widehat{=}\; !\uparrow(dtype, aname).$$
$$\left\{ \begin{array}{l} \texttt{has}(dtype)?dtype{\downarrow}\langle\rangle.\ dtype{\downarrow}(reply).\ \uparrow\langle reply, aname\rangle.\ \mathbf{0} \mid \\ \neg\texttt{has}(dtype)?\uparrow\langle NULL, aname\rangle.\ \uparrow(content, dtype). \\ dtype{\downarrow}\langle content\rangle.dtype{\downarrow}().\ \uparrow\langle ACK\rangle.\mathbf{0} \end{array} \right\} \tag{5.10}$$

5.3.2.6 *ISi* Ambient

Here we give the behaviour of the infostation in regards to the requests and communications with the ISC and the inside ambients, and formalise the behaviour in `CCA`. We will categorise the behaviour of the IS according to the inside components which send requests and receive replies.

$AAAreq_i$ With this ambient the IS behaves as follows: the InfoStation accordingly receives a request from the $AAAreq_i$ ambient, forwards it to the InfoStation Centre, so in this case the IS is just acting as a hub that receives and immediately sends. After receiving the reply from the InfoStation Centre it forwards it to the $AAAreq_i$ ambient, eventually. The IS first looks if the user device is still within its range, if yes, then it sends the reply to it, but if the user is out of its range, then it does nothing. This is because if the user device will be in the same IS after a while or any other IS in the system, then it needs send an AAA request all over again. This behaviour is modelled as:

$$\left\{ \begin{array}{l} !AAAreq_i{\downarrow}(uid, dtype, aname).ISC :: \langle AAAreq, uid, dtype, aname, IS_i\rangle.\mathbf{0} \mid \\ !ISC :: (ack, aname, slist).(\texttt{has}(aname))?\ AAAreqi{\downarrow}\langle ack, aname, slist\rangle.\mathbf{0} \end{array} \right\} \tag{5.11}$$

Lectreq$_i$ **and** *Testreq*$_i$ With the *Lectreq*$_i$ / *Testreq*$_i$ ambient, the infostation behaviour will be as the following: first, it receives a request from *Lectreq*$_i$ / *Testreq*$_i$, then it checks the availability of the requested lecture / test in its cache by sending a request to the *Cache*$_i$ ambient, i.e.

$$!Servicereq_i{\downarrow}(serviceid, uid, dtype, aname).\ Cache_i{\downarrow}\langle serviceid, uid, dtype, aname\rangle.\mathbf{0} \tag{5.12}$$

Where *Servicereq*$_i$ and *serviceid* means either *Lectreq*$_i$ and *lectid*, or *Testreq*$_i$ and *testid* respectively.

Here, we model a flag to be set by the *Cache*$_i$ ambient to a specific value (0 or 1). If the flag is set to 0, that means that the requested lecture / test does not exist in the *Cache*$_i$ ambient. But if the flag is set to 1, then the requested lecture / test exists in the *Cache*$_i$ ambient. After this stage, the IS will get a reply from the *Cache*$_i$ ambient, two possible scenarios apply for that reply; flag set to 0 or 1, as follows:

- *Flag set to* 1, which means that the *Cache*$_i$ ambient replied with the content of the lecture / test, so the IS should send a request to the ISC wit the flag set to 1, this means that the ISC only have to check if the user is using the mTest service or not, if the user is taking a mTest, then the request for a lecture / test must be denied (sending a $DENIED$ message to the user),

- *Flag set to* 0, which means that the *Cache*$_i$ ambient replied with NULL as lecture's / test's content. In this case, the IS will send a request to the InfoStation Centre with the flag set to 0, asking for both the requested lecture / test, and to check whether the user is taking a mTest. After receiving the reply from the ISC, the IS will forward a copy of the requested lecture / test to the *Cache*$_i$ ambient to store. And like the previous behaviour, if the user is using the mTest, then a $DENIED$ message will be forwarded to the user

device.

The above behaviour of the IS is modelled as follows, where $Servicereq_i$ and $serviceid$ means either $Lectreq_i$ and $lectid$, or $Testreq_i$ and $testid$ respectively:

$$!Cache_i{\downarrow}(serviceid, uid, dtype, creply, aname).$$
$$\left\{ \begin{array}{l} \neg(creply = NULL)?\ ISC :: \langle serviceid, uid, dtype, aname, 1\rangle.\mathbf{C} \mid \\ (creply = NULL)?\ ISC :: \langle serviceid, uid, dtype, aname, 0\rangle.\mathbf{N} \end{array} \right\} \quad (5.13)$$

where $\mathbf{C}$ and $\mathbf{N}$ are defined as follows:

$$\mathbf{C} \ \widehat{=}\ ISC :: (serviceid, reply, aname).$$
$$\left\{ \begin{array}{l} (reply = OK \wedge \texttt{has}(aname))?\ Servicereq_i{\downarrow}\langle serviceid, creply, aname\rangle.\mathbf{0} \mid \\ (reply = OK \wedge \neg\texttt{has}(aname))?\ ISC :: \langle serviceid, uid, dtype, aname, 0\rangle.\mathbf{0} \mid \\ (reply = DENIED \wedge \texttt{has}(aname))?\ Servicereq_i{\downarrow}\langle serviceid, reply, aname\rangle.\mathbf{0} \end{array} \right\}$$

and

$$\mathbf{N} \ \widehat{=}\ ISC :: (serviceid, aname, reply).$$
$$\left\{ \begin{array}{l} (\neg(reply = DENIED) \wedge \texttt{has}(aname))?\ Cache_i{\downarrow}\langle serviceid, reply, dtype\rangle. \\ Servicereq_i{\downarrow}\langle serviceid, reply, aname\rangle.\ Cache_i{\downarrow}(ack).\mathbf{0} \mid \\ (\neg(reply = DENIED) \wedge \neg\texttt{has}(aname))?\ Cache_i{\downarrow}\langle serviceid, reply, dtype\rangle. \\ ISC :: \langle serviceid, uid, dtype, aname, 0\rangle.\ Cache_i{\downarrow}(ack).\mathbf{0} \mid \\ (reply = DENIED \wedge \texttt{has}(aname))?\ Servicereq_i{\downarrow}\langle serviceid, reply, aname\rangle.\mathbf{0} \end{array} \right\}$$

IMNreq$_i$ Here, the infostation behaviour will be as follows: it receives a request from the *IMNreq$_i$* ambient and first it checks whether the recipient is available in its range or not, then it acts. This piece of the model represents the intelligence part of the IMN service, in which the infostation checks the location of the recipient first rather than forwarding the request directly to the ISC.

$$!IMNreq_i{\downarrow}(msg, rid, raname, sid, saname).\left\{\begin{array}{l}\neg(\texttt{has}(raname))?\ A1\ |\\ \texttt{has}(raname)?\ A2\end{array}\right\} \quad (5.14)$$

Where A1 is the act in case the recipient is not in the range of the same infostation as the sender. So, the IS forwards the request as received to the ISC to deal with it, as follows:

$$A1\ \widehat{=}\ ISC :: \langle msg, rid, raname, sid, saname\rangle.\mathbf{0}$$

And A2 is the act in case the recipient does exist in the range of the same infostation. Where the infostation first sends two requests sequentially to the ISC to check whether the sender or the recipient are using the mTest service; if either of them is taking a mTest, then the IS sends a DENIED message back to the *IMNreq$_i$* ambient, if neither is taking a mTest, the IS delivers the message to the recipient. This process is modelled as follows:

$$A2\ \widehat{=}\ ISC :: \langle Utest, sid, saname\rangle.ISC :: (reply, saname).$$
$$\left\{\begin{array}{l}(reply = DENIED)?\ IMNreq_i{\downarrow}\langle reply, rid, saname\rangle.\mathbf{0}\ |\\ \neg(reply = DENIED)?\ ISC :: \langle Utest, rid, raname\rangle.ISC :: (reply, raname).\\ \left\{\begin{array}{l}(reply = DENIED)?\ IMNreq_i{\downarrow}\langle reply, rid, saname\rangle.\mathbf{0}\ |\\ \neg(reply = DENIED)?\ IMNreq_i{\downarrow}\langle msg, sid, raname\rangle.\mathbf{0}\end{array}\right\}\end{array}\right\}$$

Eventually, the infostation receives a request from the ISC to deliver a reply ($DENIED$ or ACK) to a sender, or a message to a recipient. In this case, the IS just acts as a router and passes the received parameters to the $IMNreq_i$ ambient. This process is modelled as follows, where the parameters (a, b, c) are the alternatives of the variables $(reply, rid, saname)$ or $(msg, sid, raname)$.

$$!ISC :: (a, b, c).IMNreq_i{\downarrow}\langle a, b, c\rangle.\mathbf{0} \tag{5.15}$$

Also, the infostation may receive a reply from the $IMNreq_i$ and forwards it to the ISC, as follows:

$$!IMNreq_i{\downarrow}(ack, sid, raname).ISC :: \langle ack, sid, raname\rangle.\mathbf{0} \tag{5.16}$$

ISi After giving the behaviour of the IS separately with each component, we will combine the set of behaviours of the IS to get its complete behaviour. So, the overall infostation's behaviour can be summarised as the following:

$$P_{IS_i} \mathrel{\widehat{=}} \text{Eq. (5.11)} \mid \text{Eq. (5.12)} \mid \text{Eq. (5.13)} \mid \text{Eq. (5.14)} \mid \text{Eq. (5.15)} \mid \text{Eq. (5.16)}$$

5.3.3 Infostation Centre

Here we will give the formal specification of the ISC and the components inside (i.e. child ambients). A model of the ISC is depicted below in Figure 5.4. It encompasses ambients modelling users' accounts named after each user's ID (e.g. 301, 302 in Figure 5.4); an ambient named *Lectures* that contains all the lecture ambients, each named after the corresponding lecture ID (e.g. Lect001 and Lect002 in Figure 5.4). Each lecture ambient contains three persistent memory cells named *Phone*, *PDA* and *PC*; each storing the lecture's version suitable for the corresponding type of device. The last child ambient inside the ISC is an ambient named *Tests* that

contains all the test ambients, each named after the corresponding test ID (e.g. Test101 and Test102 in Figure 5.4). The rest of the inside ambients have exactly the same behaviour as the inside ambients of the lectures IDs ambients. These components of the ISC are formalised below.

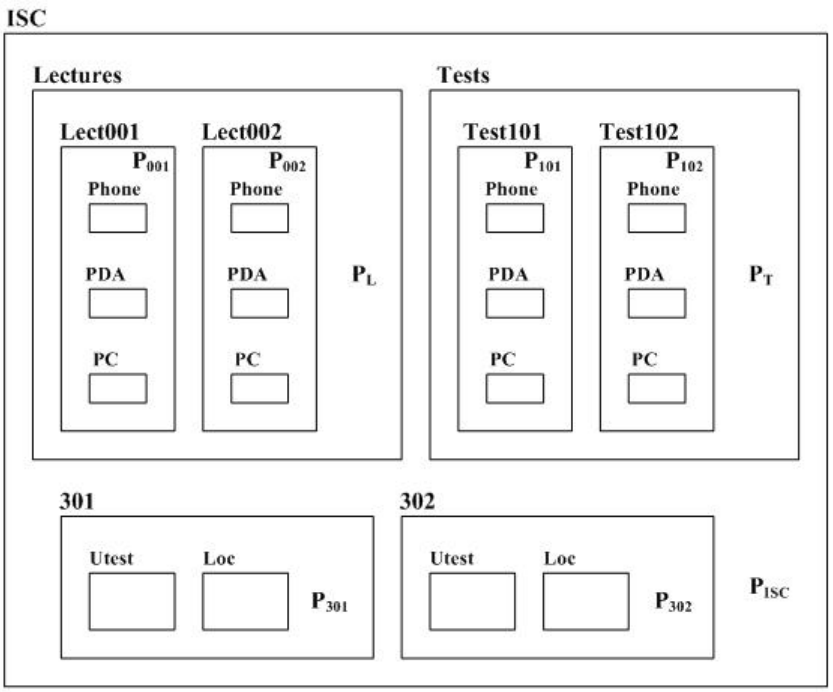


Figure 5.4: A model of the infostation centre

Users' accounts This ambient models a user's account which contains two ambients named *Loc* and *Utest*. Each of these two ambients models a persistent memory cell which stores, at any time, the current location of that user (for the former) or a Boolean indicating whether that user is taking a mTest or not (for the latter). We understand by location of a user the IS the user is registered with. The behaviour of the *Loc* ambient and the *Utest* ambient are specified exactly as in Eq. (3.1) , with appropriate initial values. The ISC requests the value of any of these cells by sending the name *Loc* or *Utest* to the user's account ambient (see Eq. (5.20)) which then can *get* (i.e. read) the value of the corresponding child ambient as follows,

where the parameter x is the corresponding child ambient name:

$$!ISC \uparrow(x).x\downarrow\langle\rangle.x\downarrow(y).\ ISC \uparrow\langle y\rangle.\mathbf{0} \tag{5.17}$$

The user's account ambient can also *put* (i.e. write) a value in any of its child ambients as follows, where the parameter x is the corresponding child ambient name and the parameter n is that value:

$$!ISC \uparrow(x,n).x\downarrow\langle n\rangle.x\downarrow().ISC \uparrow\langle x, ACK\rangle.\mathbf{0} \tag{5.18}$$

So the whole behaviour of a user's account ambient named *uid* is specified as:

$$P_{uid} \ \widehat{=}\ \text{Eq. (5.17)} \mid \text{Eq. (5.18)}.$$

Lectures / Tests As mentioned above, these ambients contain all the lectures and tests that are available in the mLecture and mTest services. Each lecture / test has a unique ID and the corresponding ambient is named after that ID. The behaviour of a lecture ambient or a test ambient is specified in Eq. (5.10). The *Lectures* ambient and the $Tests$ ambient behave exactly as the cache ambient specified in Eq. (5.9).

ISC We now formalise the behaviour of the ISC when it receives a request from an IS. When the ISC receives an AAA request from an IS, it updates the user's account with its new location and then replies to that IS with an acknowledgment along with a list of available services. For the sake of simplicity, the service list is represented by the name '$SLIST$'. This is modelled by the following process:

$$\begin{aligned} &!IS_i :: (aaareq, uid, dtype, aname).\ uid\downarrow\langle IS_i\rangle.\ uid\downarrow(ack). \\ &IS_i :: \langle ack, aname, SLIST\rangle.\mathbf{0} \end{aligned} \tag{5.19}$$

For the ISC, handling a lecture request or a test request is almost done in the same way. For a lecture request, after receiving a lecture request, the ISC checks whether the user requesting the service is currently using the mTest service. This is done by it sending a message to the corresponding user's account ambient. That user's account ambient reply with 0 for 'No' and 1 for 'Yes'. If the reply is 'Yes' then the ISC fetches the current location of the user and forward a 'DENIED' message to that location. If the reply is 'No' and the flag is set to 1, a 'OK' message is forwarded to the current user location; otherwise (i.e. reply is 'No' and the flag is set to 0), the ISC fetches the appropriate lecture version for the user device and sends it to the current user location. This behaviour is represented by the following process:

$$\begin{aligned} &!IS_i :: (lecteid, uid, dtype, aname, flag).uid{\downarrow}\langle Utest\rangle. \\ &uid{\downarrow}(y).\ (P_{ISC1} \mid P_{ISC2}) \end{aligned} \tag{5.20}$$

where

$$\begin{aligned} P_{ISC1} \ \widehat{=}\ &(y = 1)?uid{\downarrow}\langle Loc\rangle.uid{\downarrow}(z). \\ &z :: \langle lectid, DENIED, aname\rangle.\mathbf{0} \end{aligned}$$

and

$$P_{ISC2} \ \widehat{=}\ \left\{ \begin{array}{l} (y = 0 \wedge flag = 1)?uid{\downarrow}\langle Loc\rangle.uid{\downarrow}(z).\ z :: \langle lectid, OK, aname\rangle.\mathbf{0} \mid \\ (y = 0 \wedge flag = 0)?Lectures{\downarrow}\langle lectid, uid, dtype, aname\rangle. \\ Lectures{\downarrow}(lectid, uid, dtype, reply, aname).uid{\downarrow}\langle Loc\rangle.uid{\downarrow}(z). \\ z :: \langle lectid, reply, aname\rangle.\mathbf{0} \end{array} \right\}$$

Handling a test request is done in the same way as handling a lecture test apart from one additional process, which is, if a user requests a test and the user is not

using the mTest service, the ISC will deliver the requested test material, but then, it must change the value of the *Utest* ambient inside the corresponding user account from **0** to 1. This process can be specified as follows:

$$\begin{aligned} &!IS_i :: (testeid, uid, dtype, aname, flag).uid{\downarrow}\langle Utest\rangle. \\ &uid{\downarrow}(y).\ (P_{ISC3} \mid P_{ISC4}) \end{aligned} \tag{5.21}$$

where

$$\begin{aligned} P_{ISC3} \ \widehat{=}\ & (y = 1)?uid{\downarrow}\langle Loc\rangle.uid{\downarrow}(z). \\ & z :: \langle testid, DENIED, aname\rangle.\mathbf{0} \end{aligned}$$

and

$$P_{ISC4} \ \widehat{=}\ \left\{ \begin{array}{l} (y = 0 \wedge flag = 1)?uid{\downarrow}\langle Loc\rangle.uid{\downarrow}(z).\ z :: \langle testid, OK, aname\rangle. \\ uid{\downarrow}\langle Loc, 1\rangle.\mathbf{0} \mid \\ (y = 0 \wedge flag = 0)?Tests{\downarrow}\langle testid, uid, dtype, aname\rangle. \\ Tests{\downarrow}(testid, uid, dtype, reply, aname).uid{\downarrow}\langle Loc\rangle.uid{\downarrow}(z). \\ z :: \langle testid, reply, aname\rangle.\ uid{\downarrow}\langle Loc, 1\rangle.\mathbf{0} \end{array} \right\}$$

However, the ISC may also receive an IMN request from an infostation to deliver a message to a recipient. The ISC will first check if the sender is using the mTest service, if yes, it will send a DENIED message to the sender. If the sender is not using the mTest service, the ISC checks if the recipient does, if yes, it sends the sender a DENIED message. However, if neither of them is using the mTest service, the ISC checks the location of the recipient and sends him the message, and after receiving an acknowledgment, it sends an acknowledgment back to the sender as well after checking his location.

$$!ISi :: (msg, rid, raname, sid, saname).\ sid{\downarrow}\langle Utest\rangle.\ sid{\downarrow}(y_{sid}).\ (I1 \mid I2) \tag{5.22}$$

Where

$$I1 \ \widehat{=}\ (y_{sid} = 1)?\ sid{\downarrow}\langle Loc\rangle.\ \mathbf{0} \mid sid{\downarrow}(z_{sid}).\ z_{sid} :: \langle DENIED, raname\rangle.\ \mathbf{0}$$

and

$$I2 \ \widehat{=}\ (y_{sid} = 0)?\ rid{\downarrow}\langle Utest\rangle.\ \mathbf{0} \mid rid{\downarrow}(y_{rid}).$$
$$\left\{\begin{array}{l} (y_{rid} = 1)?\ sid{\downarrow}\langle Loc\rangle.\ \mathbf{0} \mid sid{\downarrow}(z_{sid}).\ z_{sid} :: \langle DENIED, raname\rangle.\mathbf{0} \mid \\ (y_{rid} = 0)?\ rid{\downarrow}\langle Loc\rangle.\ \mathbf{0} \mid rid{\downarrow}(z_{rid}).\ z_{rid} :: \langle msg, sid, raname\rangle.\mathbf{0} \mid \\ \qquad z_{rid} :: (ACK, sid, raname).\ sid{\downarrow}\langle Loc\rangle.\ \mathbf{0} \mid sid{\downarrow}(z_{sid}). \\ \qquad z_{sid} :: \langle ACK, sid, raname\rangle.\mathbf{0} \end{array}\right\}$$

As mentioned earlier that the recipient end of the IMN service may be in the same infostation range as the sender, in that case, the ISC may receive a request to validate the sender and the recipient statuses (i.e. if any of them is using the mTest service), as follows:

$$!IS_i :: (Utest, uid, aname).uid{\downarrow}\langle Utest\rangle.uid{\downarrow}(y).$$
$$\left\{\begin{array}{l} (y = 1)?\ IS_i :: \langle DENIED, aname\rangle.\mathbf{0} \mid \\ (y = 0)?\ IS_i :: \langle OK, aname\rangle.\mathbf{0} \end{array}\right\} \tag{5.23}$$

So the whole behaviour of the ISC is modelled by the following process:

$$P_{ISC} \ \widehat{=}\ \text{Eq. (5.19)} \mid \text{Eq. (5.20)} \mid \text{Eq. (5.21)} \mid \text{Eq. (5.22)} \mid \text{Eq. (5.23)}$$

5.3.4 Mobile Devices

As mentioned earlier in this chapter, there are three types of devices that we modelled the system according to; Phone, PDA and PC. Figure 5.2 depicts a graphical representation showing how mobile devices can be present inside the ISs of the mLearning system.

A mobile device works as follows, it enters an IS range and the user's device automatically sends an AAA request to the corresponding ambient. There is one restriction on that policy regarding the mobility capability, we modelled that once a user sends an AAA request, he can not move out of the IS unless receiving the reply for that AAA request. The process is modelled as the following, where the parameter *aname* is the ambient name which represents the name of the mobile device sending that request (e.g. Phone301, PDA302).

$$P_{aname} \;\widehat{=}\; !\mathtt{at}(IS_i)?\ AAAreq_i :: \langle uid, dtype, aname\rangle.\ AAAreq_i :: (ack, slist).\ P.\ \mathbf{0}$$

The process P is modelled as follows, where the parameters *amb* and *serviceid* denote $Lectreq_i$ and *lectid* OR $Testreq_i$ and $testreq_i$ respectively.

$$amb :: \langle serviceid, uid, dtype, aname\rangle \mid IMNreq_i :: \langle msg, rid, raname, sid, saname\rangle \mid M \mid \mathbf{0}$$

This process means that whenever the mobile device is at an IS, it sends an AAA request, waits for the reply then it can send a lecture/test/message request, perform a capability M or terminate.

5.4 Realisation

This section sheds the light on an important issue, which is the realisation of our policy-based architecture. The policy-based architecture has been introduced along with its components (Chapter 4). Then a case study of an mLearning system has been chosen, its architecture and policies have been presented as well (Chapter **??**). And earlier in this chapter, a model for the infostation-based mLearning system has been proposed and the behaviour of its component has been formalised. Now comes the part of the realisation, which is done by demonstrating how the model of the infostation-based mLearning system *matches* our policy-based architecture.

Now, we demonstrate the process of mapping the components of the mLearning system model onto the components of our policy-based architecture. Figure 4.3 shows the policy-based architecture, and Figures 5.5 depicts the model of the infostation-based mLearning system.

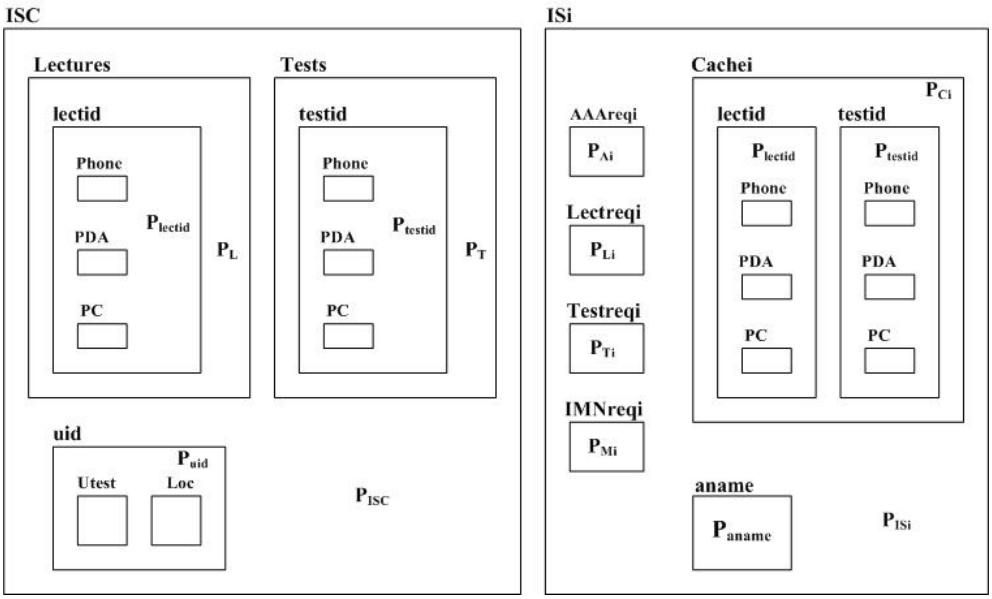


Figure 5.5: The infostation-based mLearning system's model

As mentioned in a previous chapter, the infostation-based system consists of three

tiers; ISC, ISs and user devices. The components of the system will be mapped onto the main components of the policy-based architecture which are: sensors, actuators, context interpreter, context and policy manager, context and policy database, and controller.

1. In the policy-based architecture, the *sensors* and the *actuators* represents the physical components of the system. They match the user device denoted by the '*aname*' ambient.

2. Also, the *context interpreter* matches the user device. The user device in our model is responsible for generating a record which contains the right information to each relevant request.

3. The *context and policy manager* in our architecture matches the ISC and the IS. The ISC and the ISs are responsible for managing the system, as there processes show the way of how they interact with the different components of the system to provide the right context information and how they represent the policies of the system.

4. The *context and policy database* in our architecture matches the $Cache$ ambient in the IS and the $Lectures$ and $Tests$ ambients in the ISC.

5. The *controller* in the policy-based architecture matches the processes of the ISC and the IS. These processes define the way of how to take decisions and enforce them.

5.5 Summary

In this chapter, we have introduced an ambient-based model for the infostation-based mLearning system, and then formalised the ambients of the system based on

the policies of some of the mServices, which are: AAA, $mLeacture$, $mTest$ and IMN. However, to realise our policy-based architecture, we then mapped the ambients/components of the system's model onto the components of our architecture. The realisation process is demonstrated to show that our proposed architecture can be implemented in real world.

In the next chapter, we are going to present the execution environment of the `CCA`; `ccaPL`, and execute the `CCA` processes of mLearning system in order to prove two properties, as a proof of concept.

Chapter 6

Tool Support

Objectives:

- Present the syntax and context expressions of `ccaPL`
- Present the `ccaPL` execution environment
- Develop an animator for `ccaPL` programs
- Use `ccaPL` to validate some properties of the infostation mLearning system

6.1 ccaPL: A Programming Language for CCA

The mathematical notation of CCA was presented previously in chapter (3). We showed that CCA is a suitable notation to model mobile systems that are context-aware. We also have demonstrated some examples of how to model using CCA. The syntax and semantics of CCA were given and explained with examples. ccaPL is an executable format of the CCA syntax which has been proposed by Siewe *et al.* [96]. In this chapter, we will show the syntax and the execution environment of ccaPL.

6.1.1 Syntax of ccaPL

It is hard to reproduce the syntax of CCA using a text editor (e.g. *notepad* on Windows OS, *emax* on Linux OS) for there are special symbols in the syntax of CCA such as: '↑' and '↓'. Thus, we will present the syntax of ccaPL which is a computer readable version of the syntax of CCA. Tables (6.1, 6.2, 6.3) show the syntax of ccaPL in terms of processes, capabilities and location.

We denote by h the translation function from CCA to ccaPL; that is for P in a CCA process, $h(P)$ is the corresponding ccaPL program.

6.1.2 Context Expressions in ccaPL

The following table (Table 6.4) shows the context expressions of ccaPL. Like the syntax of ccaPL, the symbols used in CCA were not easy to reproduce in a regular text editor tool, instead, ccaPL uses the meaning of those symbols to represent the context expressions in a natural fashion.

Table 6.1: Capabilities of ccaPL

Capabilities		
CCA M	ccaPL $h(M)$	**Description**
in n	in n	move into the ambient n
out	out	move out
del n	del n	delete ambient n
$\alpha\ x\langle\tilde{z}\rangle$	$h(\alpha)\ x(\tilde{z})$	call to the process abstraction x
$\alpha\ (\tilde{y})$	$h(\alpha)$ recv$(\tilde{y})$	receive list of messages $\tilde{y}$ from α
$\alpha\ \langle\tilde{y}\rangle$	$h(\alpha)$ send$(\tilde{z})$	send list of messages $\tilde{z}$ to α

Table 6.2: Processes of ccaPL

Processes		
CCA P	ccaPL $h(P)$	**Description**
0	**0**	inactivity
$n[P]$	$n[h(P)]$	ambient
$P\|Q$	$h(P)\|h(Q)$	parallel composition
$!P$	$!h(P)$	replication
$(\nu n)P$	new $nh(P)$	restriction
$\kappa?M.P$	$< h(\kappa) > h(M).h(P)$	context-guarded capability
$x \triangleright (\tilde{y}).P$	proc $x(\tilde{y}).h(P)$	process abstraction x
$\{P\}$	$\{h(P)\}$	brackets

6.1.3 ccaPL Execution Environment

Figure 6.1 below shows the architecture of the programming environment of ccaPL.

The editor component represents the CCA program development environment. The user is free to use any text editor (e.g. notepad) to write a CCA program. The next component is the parser which is developed and generated using JavaCC [1]. The parser is the component which is responsible to check the syntax for a written ccaPL program, that is; if there is a syntactic error in a ccaPL program, the execu-

[1] Java Compiler Compiler: is a special package of Java used to generate compilers [4]

Table 6.3: Location of ccaPL

Location		
CCA α	ccaPL $h(\alpha)$	**Description**
$\uparrow$	@	any parent
$n\uparrow$	n@	parent n
$\downarrow$	#	any child
$n\downarrow$	n#	child n
::	::	any sibling
n ::	n ::	sibling n
ϵ	ϵ	locally

tion process will not proceed. The interpreter component has also been generated using JavaCC. It is responsible for the execution of each single process of a ccaPL program based on the semantics and reduction rules of CCA. The console is part of the execution environment that is still under construction. The animator is a part of our contribution, so it will be discussed in a separate section later.

The execution of a ccaPL program should be done as follows: firstly, a ccaPL program should be written using the aforementioned syntax of ccaPL, then saved into a CCA file format (.cca). Then, it can be executed using the JavaCC generated parser and interpreter (the parser and the interpreter are implemented in a single java class). However, the body of a ccaPL program is treated as a single process. Comments can be added to a ccaPL program just like a java program, i.e. using the single line commentary symbol "//" or multiple lines commentary symbol "/*" followed by "*/" at the end of the comment. The CCA parser is case sensitive.

In order to execute a ccaPL program using the execution environment, the computer has to have a JDK (Java Development Kit) installed, a JDK 1.0 or a later

Table 6.4: Context Expressions of ccaPL

Context Expressions κ		
CCA κ	ccaPL $h(\kappa)$	**Description**
T	`true`	true
$n = m$	$n = m$	name match
$\bullet$	`this`	hole
$\neg\kappa$	`not` $h(\kappa)$	negation
$\kappa_1 \mid \kappa_2$	$h(\kappa_1) \mid h(\kappa_2)$	parallel composition
$\kappa_1 \vee \kappa_2$	$h(\kappa_1)$ `or` $h(\kappa_2)$	disjunction
$\kappa_1 \wedge \kappa_2$	$h(\kappa_1)$ `and` $h(\kappa_2)$	conjunction
$\oplus\kappa$	`next` $h(\kappa)$	spatial next modality
$\diamond\!\!\!\!+\ \kappa$	`somewhere` $h(\kappa)$	somewhere modality

version will be enough. Below is an example of a ccaPL program:

```
BEGIN_DECLS
def has(n) = this | n[true] | true
def at(n) = true | n[next(this | true) | true]
END_DECLS

//Here goes the program
IS1[ Phone123[ IS1@send(HelloParent).IS1@recv(x).out.0]
| Phone123#recv(y).<y=HelloParent>Phone123#send(HelloChild).0
 ]
```

The preamble of a ccaPL program is represented by the part starting with '`BEGIN_DECLS`' and ending with '`END_DECLS`', noting that the begin and end part should be written in capital letters. The preamble contains definitions of context expressions, in which it is presented in the following form: `def has(n) = this | n[true] | true`. The preamble of a ccaPL program is an optional section.

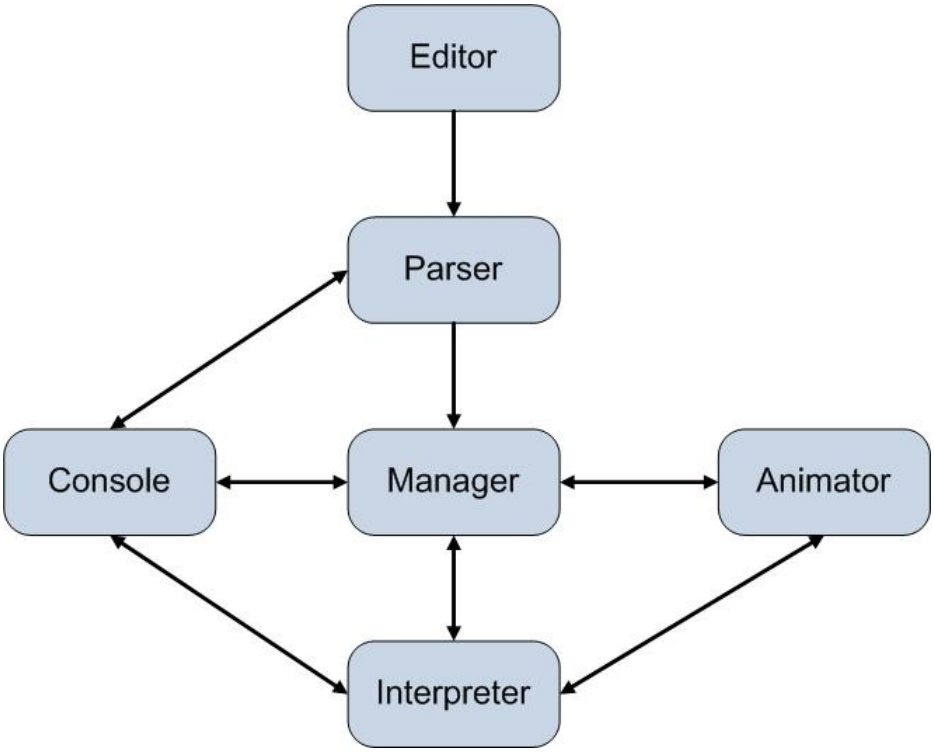
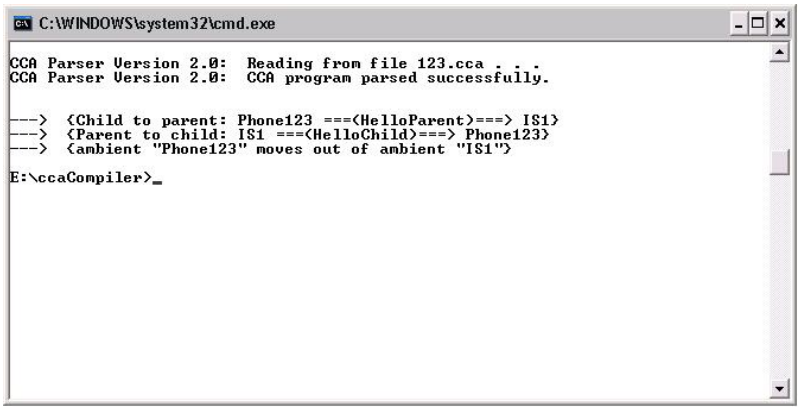
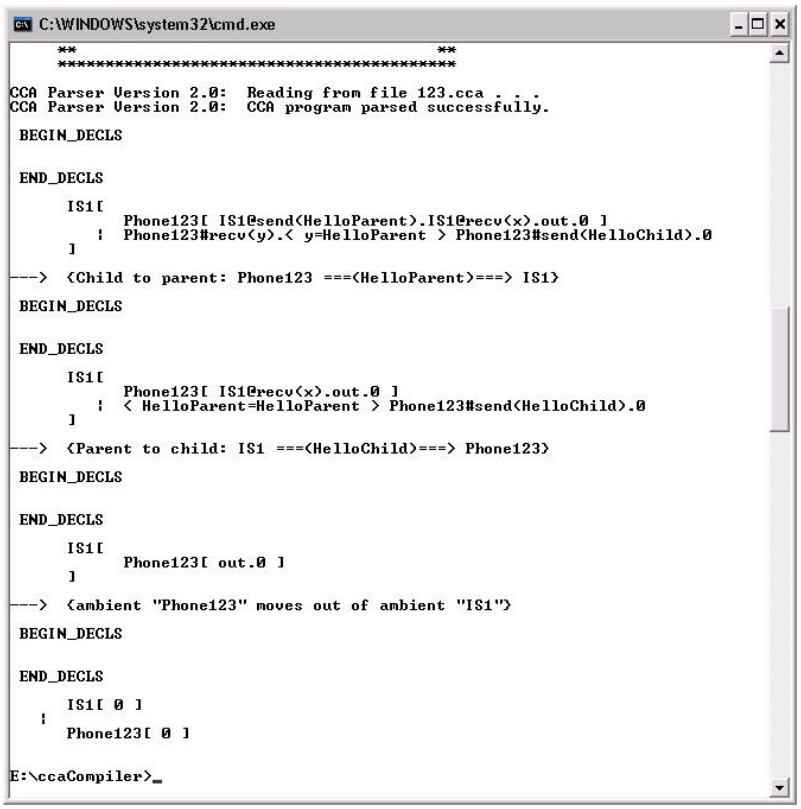


Figure 6.1: The architecture of the execution environment of `ccaPL`

In order to have a sense of what the execution environment looks like, we are going to execute the above example. Figure 6.2a shows a snapshot of the result of executing the example. This figure shows the reductions that happened by executing the processes of the `ccaPL` program. However, a little amendment can be made to the `ccaPL` program to enable the processes to be displayed in the execution along with the reduction relations, This is done by adding '`display code`' to the preamble of the program. The result of this is shown in Figure 6.2b.



(a) Reduction relations only



(b) Reduction relations and processes

Figure 6.2: A snapshot of the execution environment of `ccaPL`

Now, each single line in the output of an execution is read as follows: the symbol '`--->`' represents the reduction relation of `CCA` as formally defined in [96]. The explanation of each transition is given between a pair of curly brackets. In particular, the notation '`A ===(X)===> B`' means that an ambient '`A`' sent a message '`X`' to

ambient '`B`'. The notations '`Child to parent`', '`Parent to child`' and '`Sibling to sibling`' provide information about the relationship between the sender '`A`' and the receiver '`B`' with regards to ambient hierarchy. moreover, the notation '`Local`' means that the ambient is communicating with itself.

6.2 `ccaPL` Animator

Until this stage, it is hard to read the output of the execution environment by anyone with no prior knowledge of formalism (specifically `CCA`). So, one of our contributions throughout this thesis, is to propose an animator for the execution environment of `ccaPL`. The animator is a tool that animates the processes of a `CCA` program. It is basically a graphical representation of the ambients and their corresponding processes.

As simulating `CCA` processes is not easily readable, we propose a graphical aid for the execution environment of `ccaPL`. It facilitates understanding the ambients associated with a `CCA` program in terms of their location within their environment, and the processes associated with each ambient. The animator is a useful tool because it offers the chance to visualise:

- the transitions throughout the processes;
- how and when each ambient's capability is performed;
- and how ambients move around their surrounding environment.

For the reasons we mentioned above, and to give the mathematical notation of `CCA` a complete programming environment, it is important to have a complementary animation tool to the parser and the interpreter.

The ccaPL animator is implemented using Java (the Swing class). It is implemented as an individual class which is called by the parser/interpreter during the process of execution. Figure 6.3 below depicts a snapshot of the animation of the example in the previous section, The snapshot shows the output of the example after the final transition is executed. The main animator window is named '*University Campus*', for the example we demonstrated is of the infostation-based mLearning system which is implemented throughout a university campus.

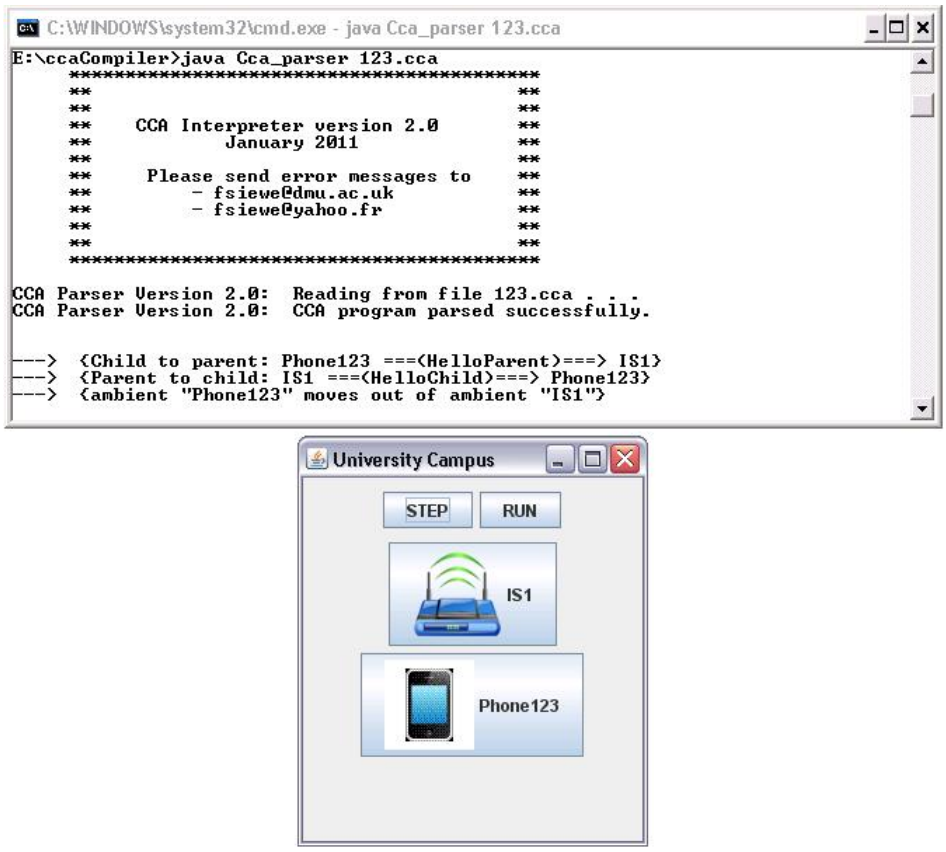


Figure 6.3: A snapshot of the execution environment of ccaPL + animator

The animator is designed in a hierarchical way, in which the root ambient (the first ancestor ambient which has no parent ambient) of the ccaPL program is represented as a container, which contains the child ambients inside. Each child ambient inside the main container is represented as a button, consists of a name and a pic-

ture/icon. The icons associated with the mLearning system are: phone, PDA, PC, router for infostation and server for infostation centre. When a user click an ambient (i.e. button), there are two possible scenarios:

- if the clicked ambient has no child ambients, then a dialogue message box appears which contains the processes of that ambient represented as a text message,
- however, if the clicked ambient has at least one child ambient, then clicking that ambient will generate another container which contains the child ambient(s) and a button which clicking shows a dialogue message box which contains the inside processes of that ambient, in a text format as well.

The inside ambients of an ambient are represented in the same way the ambient is represented (i.e. as buttons). The container of the root ambient has two buttons, initially. One is labeled 'Run', the other is 'Step'. A `CCA` program execution operation will initially execute the first transition then halts the execution and displays the animation. Afterwards, a user has a choice of hitting the 'Run' button to fully execute the program and display the animation after the final transition, or the user can simply click the 'Step' button to go to the next transition, halts the execution then display the animation as well. The coming example shows how mobility transitions are represented in the `ccaPL` animator.

Example 6.2.1 *A system consists of two infostations; $IS1$ and $IS2$, and two devices; $PC303$ and $PDA301$. $PC303$ eventually performs mobility capabilities in which it moves out of its initial location; $IS1$, and into its destination; $IS2$. Also, the different ambients of the system execute some follow up processes. The* `CCA` *representation of the system is as follows:*

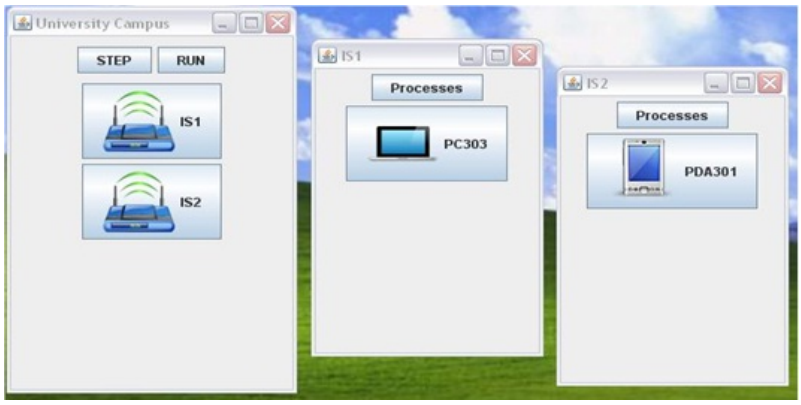
```
IS1 [
```

```
      PC303[@recv(x) . <x=2> out. in IS2. ::send(Hello).0 |
      ::recv(y).0] |
      #send(2).0 | IS2::recv(b).0 ] |
IS2 [
      PDA301[::recv(z) . <z=Hello> ::send(Welcome).
      @send(HeIsHere).0] |
      #recv(s). <s=HeIsHere> IS1::send(IHaveBoth).0 ]
```

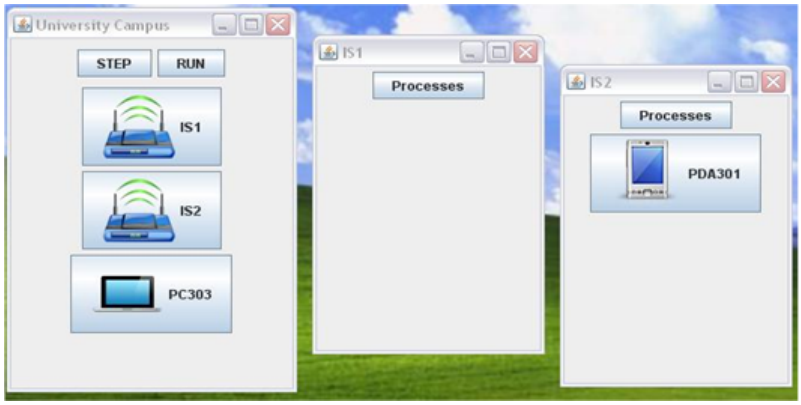
However, figure 6.4 shows three sequential figures in which they show how the device $PC303$ moves around in its environment. Figure 6.4-a depicts the initial state of the system where $PC303$ is in $IS1$. Figure 6.4-b shows $PC303$ after moving out of $IS1$ (i.e. becoming a sibling ambient to $IS1$). Figure 6.4-c shows $PC303$ after performing the capability ***in*** *$IS2$. Each transition is executed manually by a user clicking the 'Step' button.*

We showed how a mobility capability is represented in the `ccaPL` animator. Now, we show how the processes of each ambient is represented in the animator. The same system processes of the previous example are going to be used.

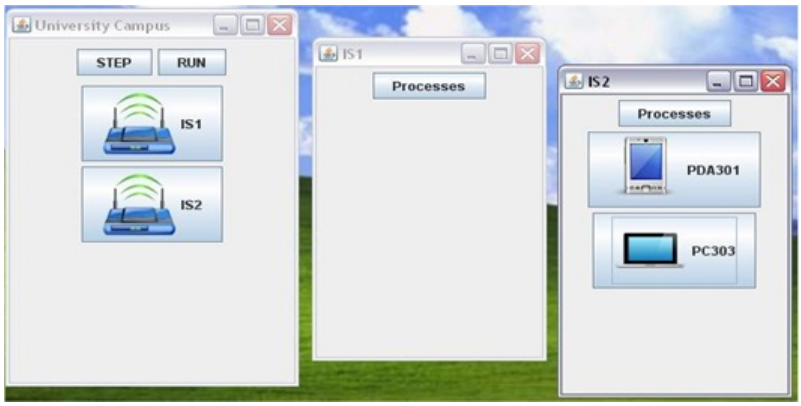
Example 6.2.2 *We will put some snap shots showing how to view the processes of an ambient in the* `ccaPL` *animator, using the same system model in the previous example. For example, to show the associated processes of an infostation, a user should click the button of the infostation, which will show a container that contains the processes and the child ambients, clicking on the 'processes' button will show the processes of the corresponding infostation. Figure 6.5 shows the processes of infostation $IS2$. However, clicking on the button of any device despite its location within the system, will show the processes of that exact ambient. Figure 6.6 show the processes of ambient $PDA301$.*



(a) Initial location of ambient $PC303$



(b) Ambient $PC303$ after performing the capability '`out`'



(c) Ambient $PC303$ after performing the capability '`in` $IS2$'

Figure 6.4: Mobility transitions

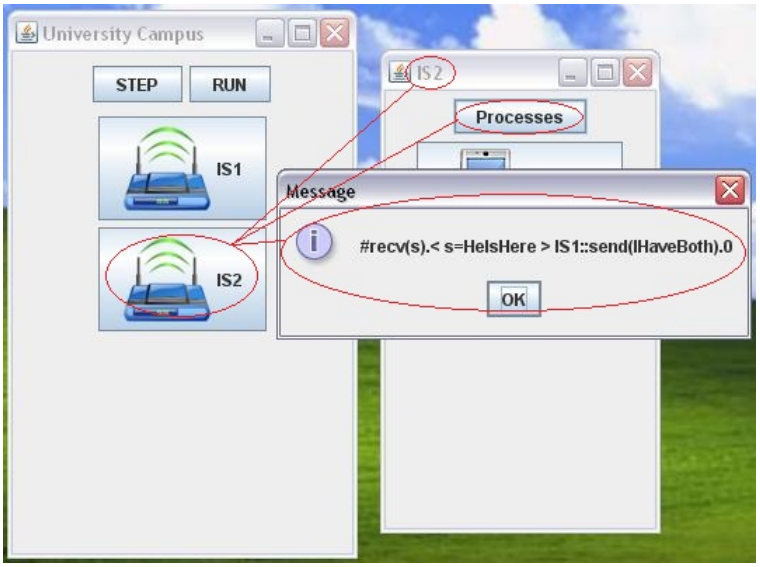


Figure 6.5: Processes of an infostation ambient

6.3 Validation

In this section, we are going to show the validation of some properties regarding the mLearning system. First, we will use the mLearning system specification that has been formalised in Chapter (5) and run all the processes of the ISC and the ISs in parallel and inject some users behaviour along with the system. However, we first must transform the specification of the mLearning system from `CCA` to `ccaPL` syntax in order to run then using the `CCA` interpreter. Moreover, every user's behaviour we will propose from now on, will be written using the `ccaPL` syntax. A property of a concurrent system is defined in terms of concept as: *an assertion that is true for every possible execution.* Lamport [77] has proposed two main classes of system properties:

Safety property: stating that nothing bad will happen.

Liveness property: stating that something good will happen, eventually.

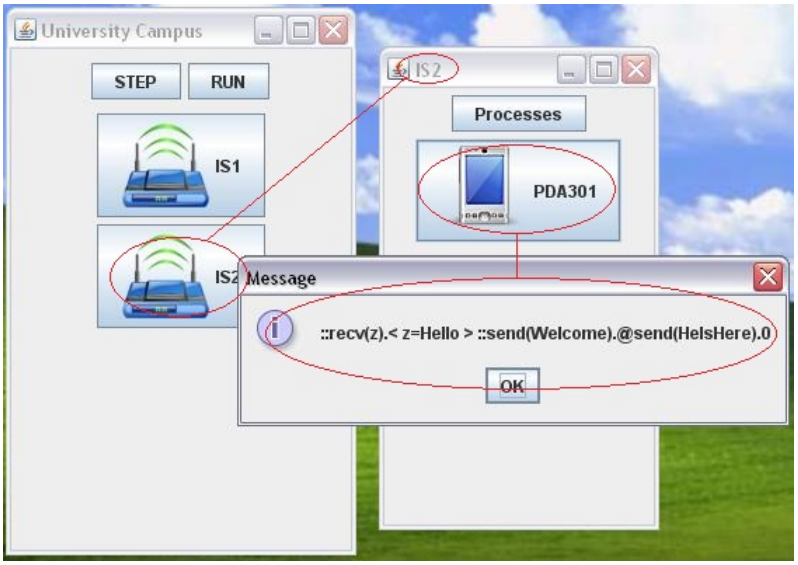


Figure 6.6: Processes of a device ambient

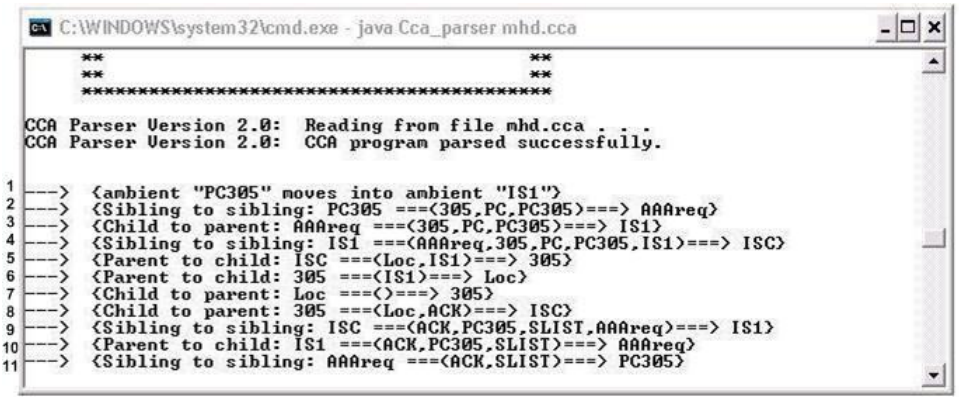
In the lights of this classification, we will validate some properties regarding the mLearning system, as well as the learning process in general. The validation process will be in the form of proposing some scenarios (i.e. user behaviour), invoke them to the mLearning system, and observe how the system will behave in return. The anticipation here is to witness the system acts correctly according to the model and the formalised policies of the infostation-based mLearning system.

The full formal executable `CCA` specification of the infostation-based mLearning system is given in Appendix C. The specification is composed of two ISs and one ISC.

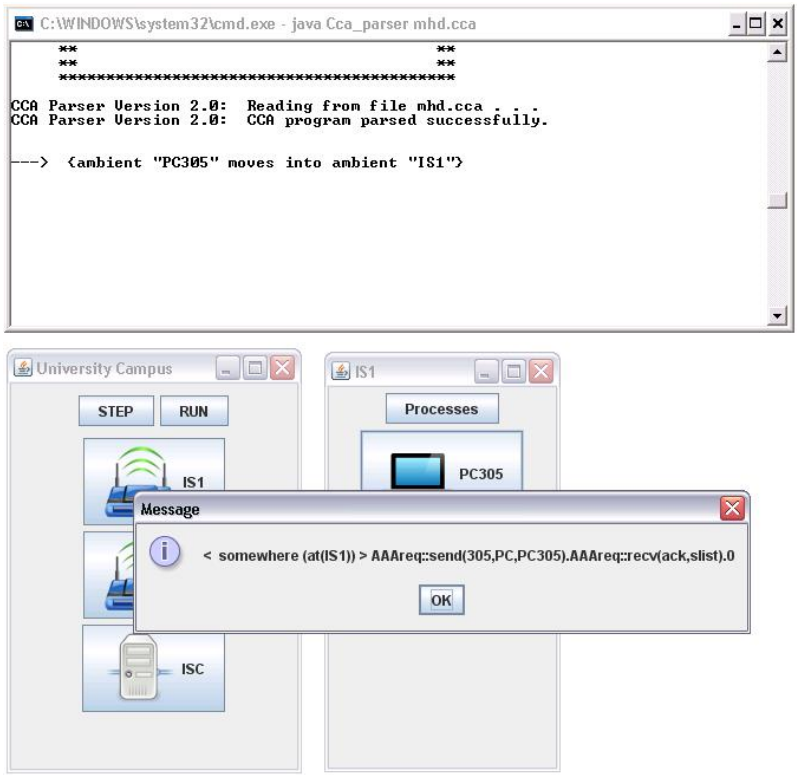
Scenario 1: This scenario validates the *AAA* service. The property we want to validate here is: "*when a user moves into an IS, the user's device registers itself automatically with that IS*". When a user moves into an IS, its device sends an AAA request to the *AAAreq* ambient and waits for a reply which contains an acknowledgment and the list of services. The user process will run in parallel with the mLearning system processes. The user process of the AAA request is as follows:

```
PC305[ in IS1. 0 |
       <somewhere at(IS1)> AAAreq :: send(305, PC, PC305).
       AAAreq :: recv(ack, slist).0
     ]
```

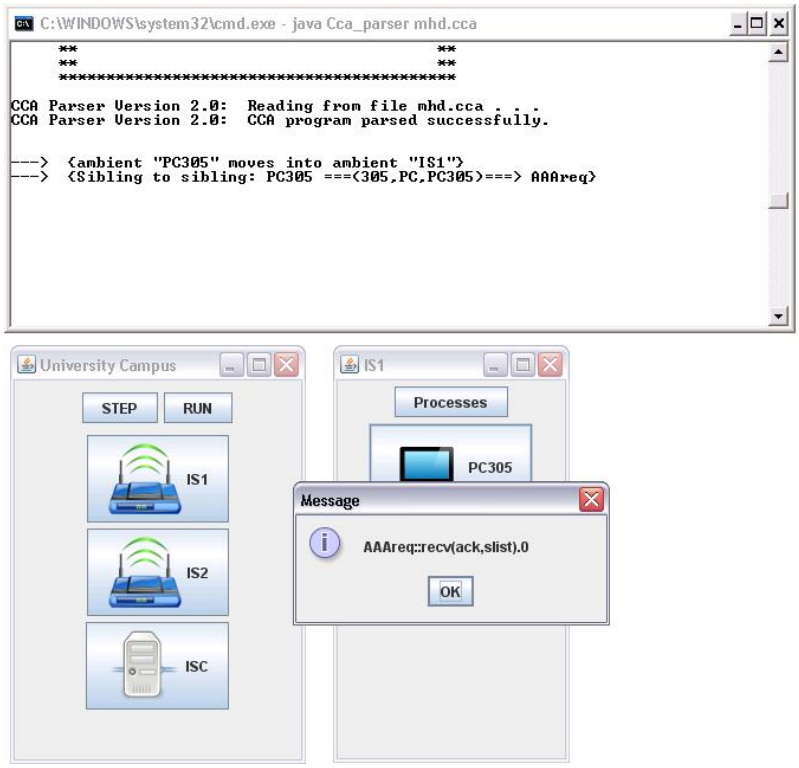
Figure 6.7 below shows a screen shot of the result of executing the above user's behaviour in parallel with the mLearning system's behaviour. It shows the reductions that took place, the ambient-to-ambient communications, and how when a user sends an AAA request, (s)he will eventually receive a reply back as a sign of him/her registering with that IS.


C:\WINDOWS\system32\cmd.exe - java Cca_parser mhd.cca
**
**
****
CCA Parser Version 2.0: Reading from file mhd.cca . . .
CCA Parser Version 2.0: CCA program parsed successfully.
1 ---> <ambient "PC305" moves into ambient "IS1">
2 ---> <Sibling to sibling: PC305 ===<305,PC,PC305)===> AAAreq>
3 ---> <Child to parent: AAAreq ===<305,PC,PC305)===> IS1>
4 ---> <Sibling to sibling: IS1 ===<AAAreq,305,PC,PC305,IS1)===> ISC>
5 ---> <Parent to child: ISC ===<Loc,IS1)===> 305>
6 ---> <Parent to child: 305 ===<IS1)===> Loc>
7 ---> <Child to parent: Loc ===<)===> 305>
8 ---> <Child to parent: 305 ===<Loc,ACK)===> ISC>
9 ---> <Sibling to sibling: ISC ===<ACK,PC305,SLIST,AAAreq)===> IS1>
10 ---> <Parent to child: IS1 ===<ACK,PC305,SLIST)===> AAAreq>
11 ---> <Sibling to sibling: AAAreq ===<ACK,SLIST)===> PC305>
```

Figure 6.7: Execution of the AAA request

The screen shot shows that when a user moves into an IS (line 1), $IS1$ in this case, the user's device sends AAA request to the *AAAreq* ambient in that IS (line 2), the *AAAreq* ambient forwards the request to the IS (line 3), which will send the request to the ISC (line 4). Then what happens inside the ISC is this, it updates the location of that user and sends an acknowledgment to the corresponding IS (lines 5-9). Finally the user receives the acknowledgment along with the list of services (line 11).

Figure 6.8a shows the execution and animation of the user and system behaviours when a user sends an AAA request. The figure shows a screen shot in which only the first step of the process is executed and animated. It shows that the user has
```

moved into $IS1$. Figure 6.8b shows the second step of the execution and animation, in which the user device has sent the AAA request and is waiting for the reply.



(a) First step



(b) Second step

Figure 6.8: Execution and animation of the AAA service

Figure 6.9 shows the last step of the execution and animation in which the user device received the reply from the *AAAreq* ambient, then terminates.

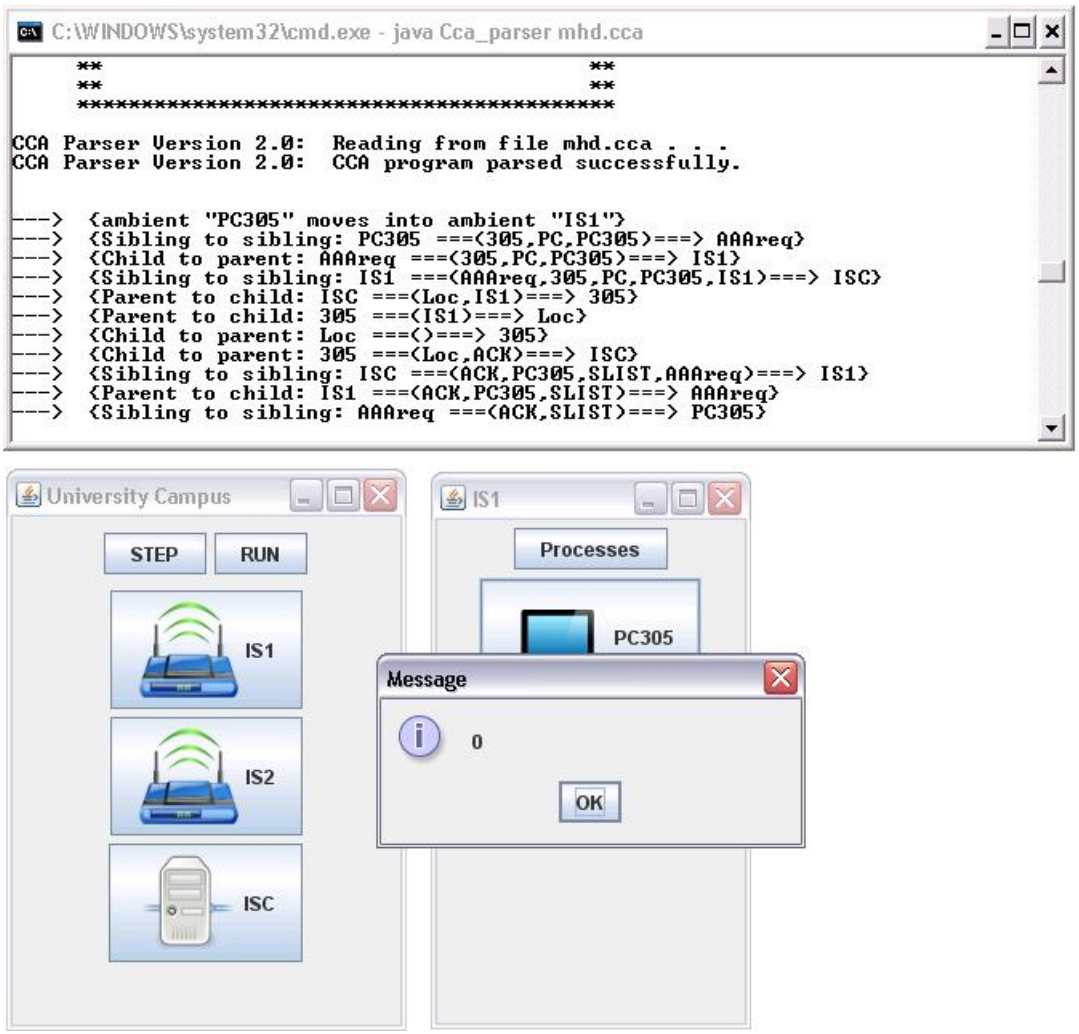


Figure 6.9: Completed execution and animation of AAA request: last step

Scenario 2: This scenario presents how the system behaves when a user requests a lecture. The property to validate here is: 'When a user requests a lecture, (s)he will eventually get a reply, providing that the user stays long enough in the system'. Suppose a user moves into IS1, register then request a lecture, say *Lect*001. We assume that the value inside the *Utest* ambient for the user is 000 initially (i.e. not using the mTest service). However, in this scenario, the *Cache* ambient of IS1 has

a copy of the requested lecture. The user requests the lecture and receives the reply in the same IS. Figure 6.10 shows the execution and animation of the user after sending the lecture request. In this scenario, we suppose that the user is already in IS1 and has registered previously.

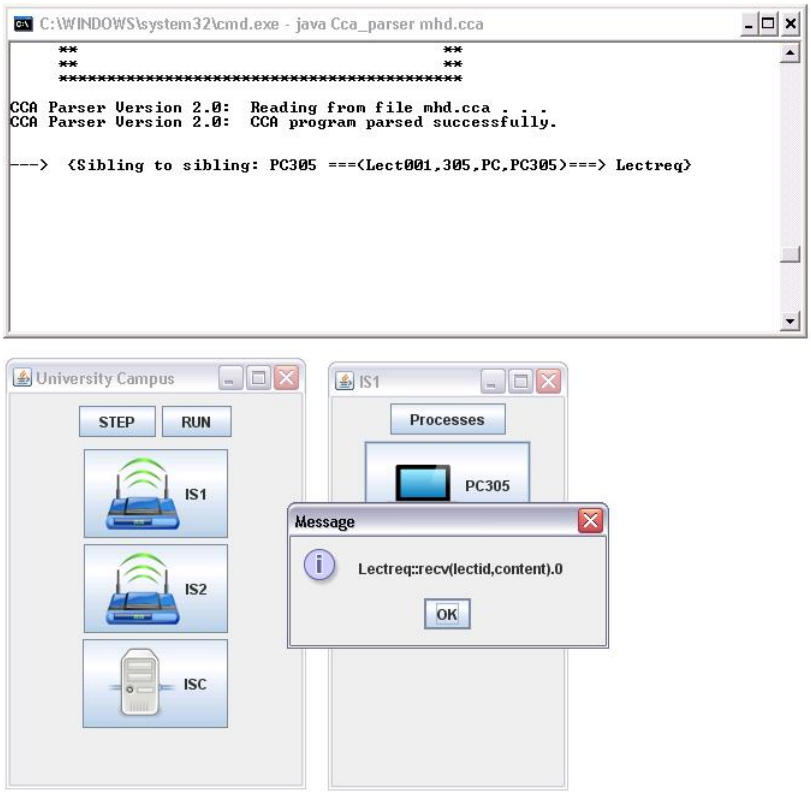


Figure 6.10: Execution and animation of mLecture service: user requested a lecture

However, Figure 6.11 shows the execution and animation after the user received the reply. *Line 2* shows that the *Lectreq* ambient has forwarded the user's request to the IS. *Lines 3-9* shows that the IS has sent a request to the *cache* ambient to check whether it has the requested lecture inside. The *cache* then finds and fetched the corresponding format of the requested lecture and forwards the content to the IS. *Line 10* shows that the IS has forwarded a message to the ISC just to check if the user is using the mTest service or not, the message contains a flag set to 1 implying that the IS has the requested lecture. *Lines 11-15* shows that the ISC has sent a request to the corresponding user's profile to check if he is using the mTest

service or not, the user's profile then replies with the value 000 meaning that the user is free to use any service, then from *line 16* to *line 21* the ISC fetches the user's location. Afterwards, the ISC sends the reply to the corresponding IS and the IS to the *Lectreq* ambient which forwards the reply to the user.

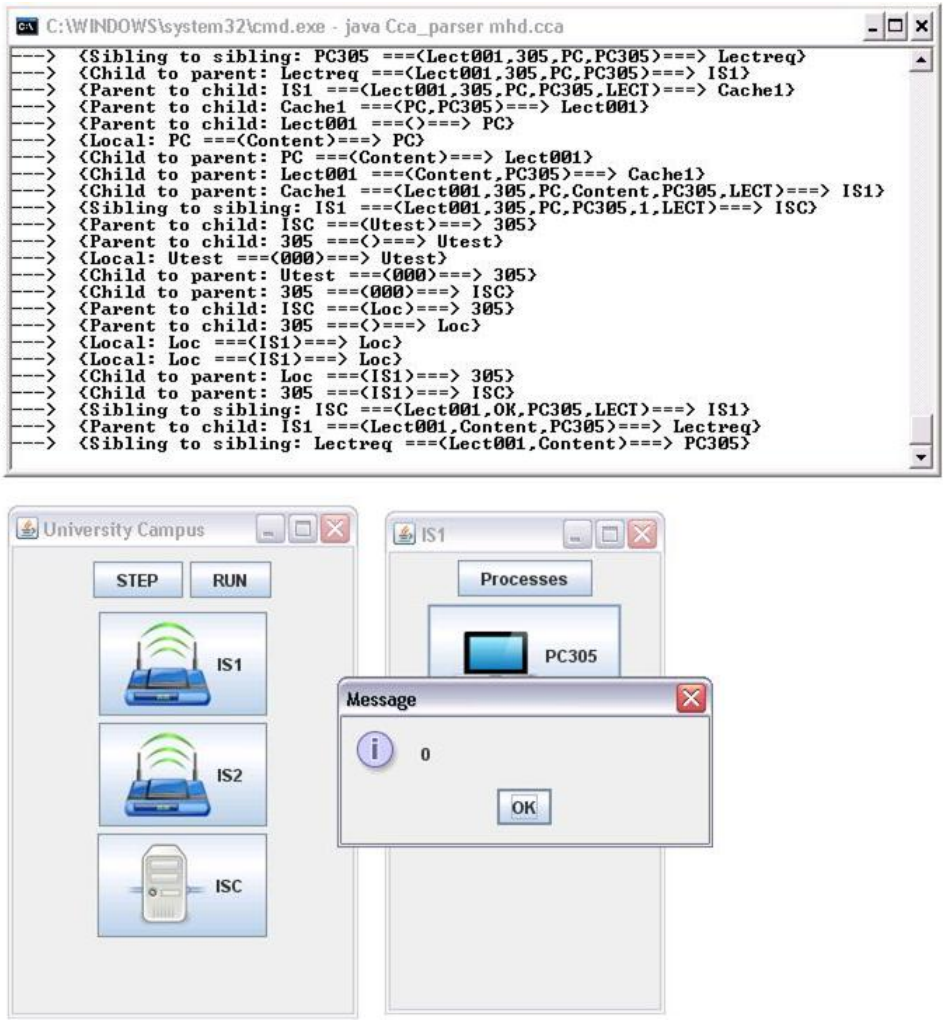


Figure 6.11: Execution and animation of mLecture service: user received the lecture

Scenario 3: This scenario validates that: "*when a user requests the mTest service, the user receives the test and his profile changes so that (s)he can not request any other service while using the mTest service, thus can not cheat*". Figure 6.12 depicts the execution and animation showing that user $PC305$ has moved into IS1, registered then requested $Test103$.

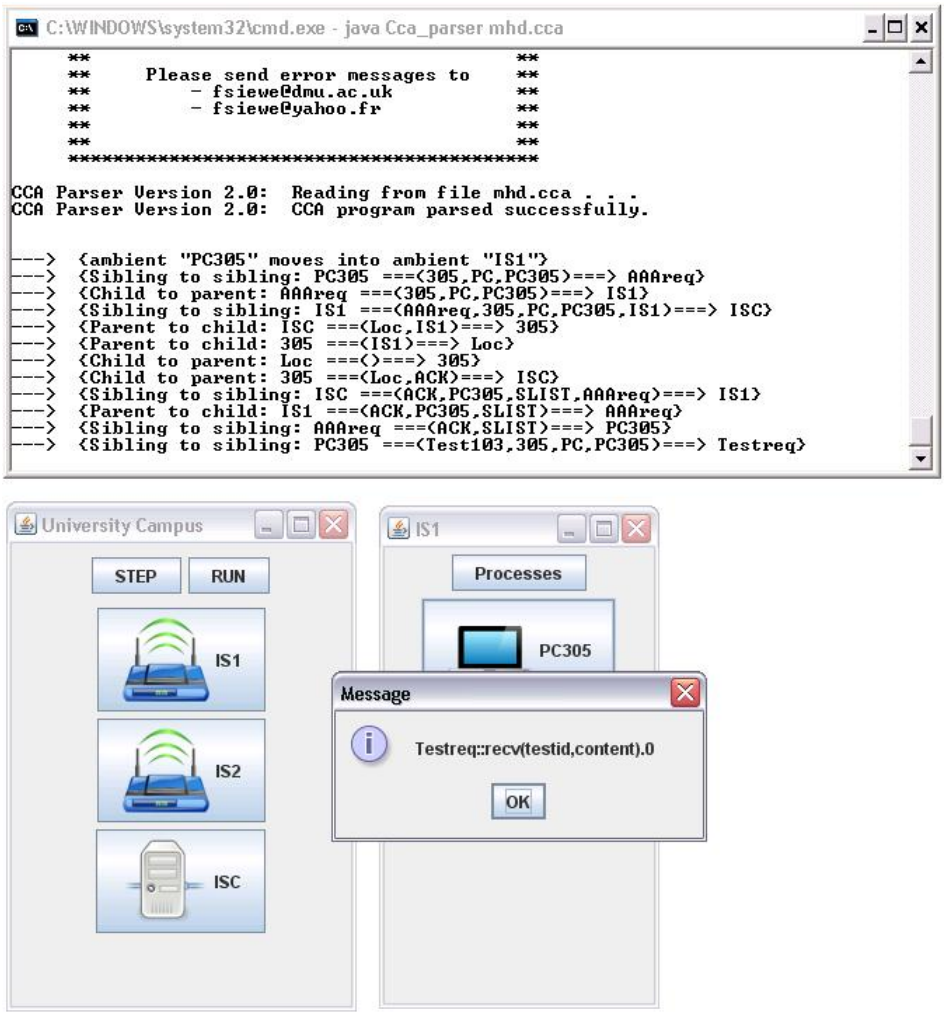


Figure 6.12: Execution and animation of mTest service: user requested a test

Figure 6.13 shows the execution and animation of the last transition happened in the whole process of requesting the mTest service. The request and delivery process is almost the same as the request and delivery of the mLecture service, the system's behaviour is exactly as it is regarding the mLecture service, however, when a user requests a test material, his/her profile should change in a way that does not allow that user to use any other service with the mTest service simultaneously. This process of manipulating the user's profile is shown in *lines 18 - 24* where the ISC has changed the value of the *Utest* ambient to 1, meaning that the user has started using the mTest service.

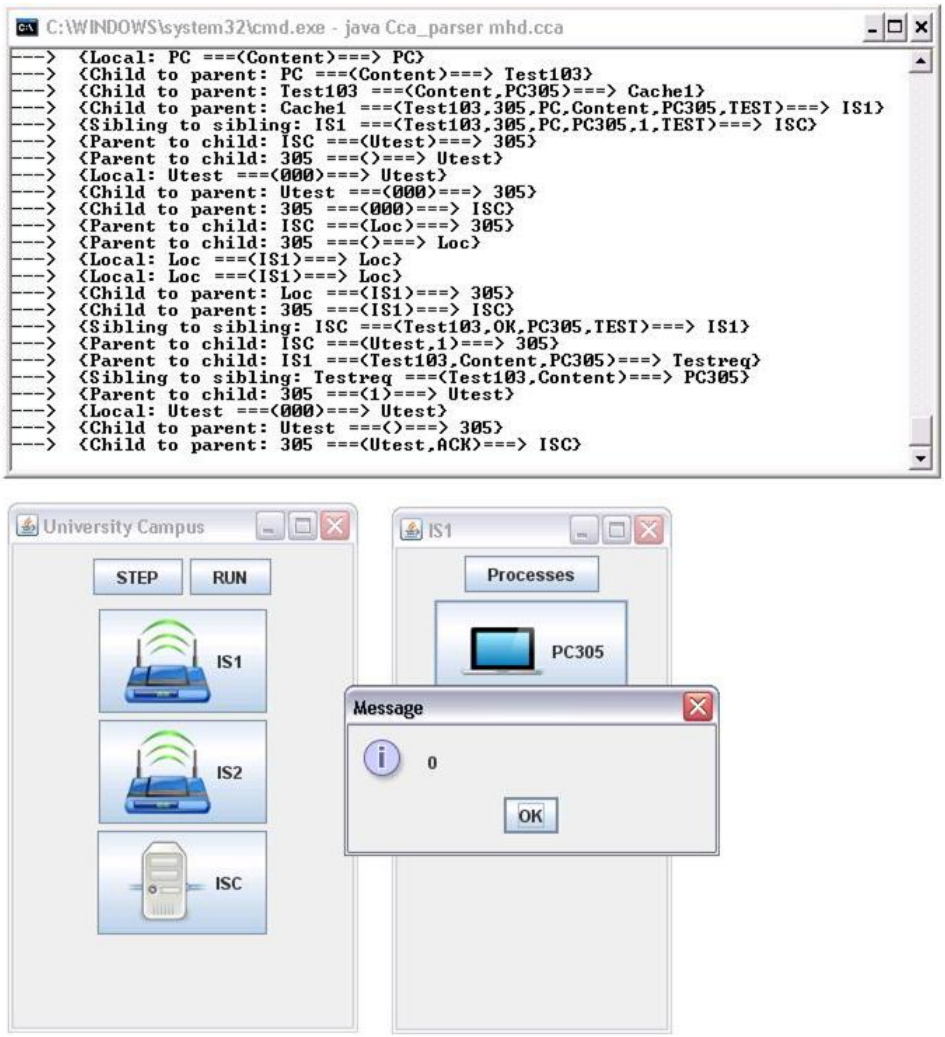


Figure 6.13: Execution and animation of mTest service: user started a test

Scenario 4: For this scenario, the user requests a lecture, however, the user is using the mTest service (i.e. *Utest* value is equal to 1). Thus, the user must receive a '*DENIED*' message. If in this case the system behaves as desired, then it is a validating to the safety property stating that "*When a user is using the mTest service, (s)he can not use any other service simultaneously*".

To validate this property, suppose user $PC305$ is using the mTest service requests $Lect002$, however, the requested lecture exists in the *Cache* ambient of IS1. The user's request should then be denied. Figure 6.14 shows the execution and the

animation of the final process in which the user receives a *DENIED* message.

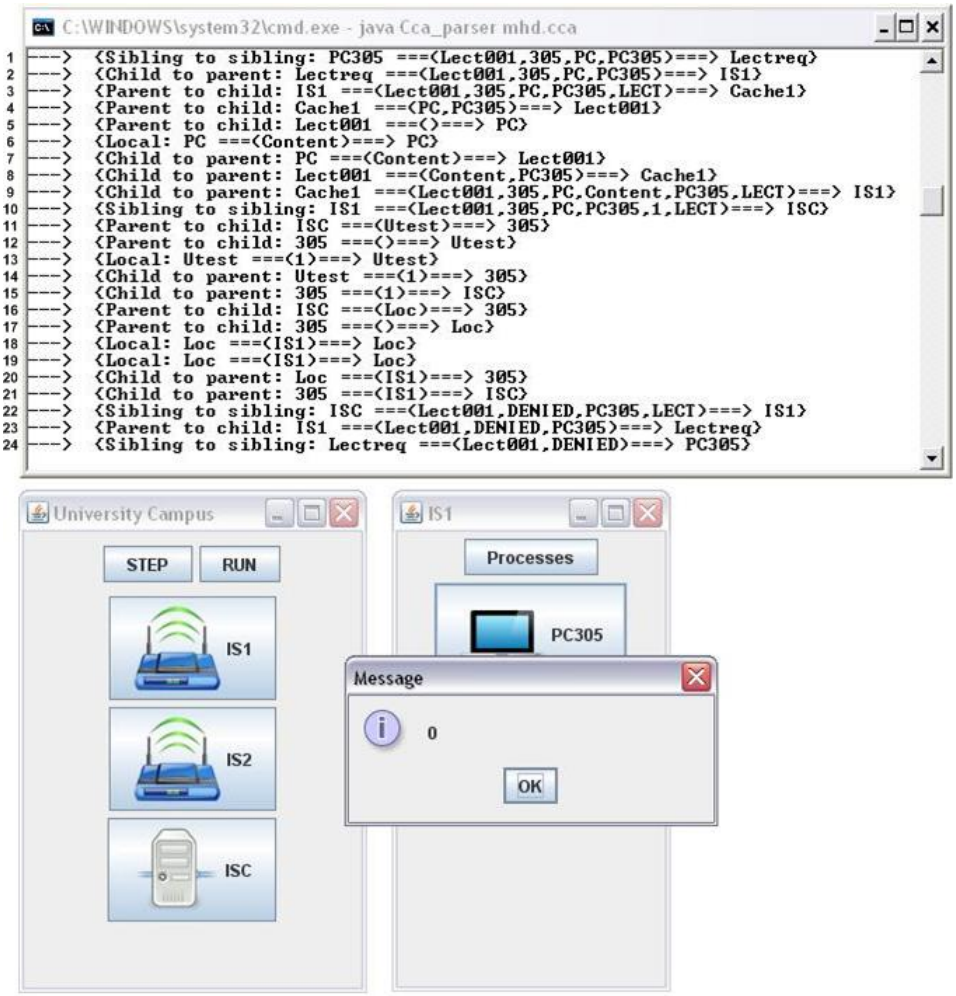


Figure 6.14: Execution and animation of lecture request: request denied

Line 10 shows that the IS is forwarding a message to the ISC to check if the user is using the mTest service with a flag set to 1 indicating that the IS has the requested lecture. *Line 13* shows that the value of the *Utest* ambient of the corresponding user is equal to 1, thus from *line 16* to *line 21* the ISC fetched the current location of the user and sends a $DENIED$ message to the corresponding IS. *Line 24* shows that user $PC305$ has received a message contains the lecture ID and a $DENIED$.

Scenario 5: This scenario validates the property that says: "*when a user sends a message to another user, the message will eventually get delivered*". This property

is valid given that the recipient side of the message stays in the system long enough.

To validate this property let us suppose the following scenario: user $PC305$ sends a *HELLO* message to user $Phone306$, where the two users are in the same IS. Initially the users' location is $IS1$. Figure 6.15 shows the execution and animation of IMN service where both users (i.e. sender and recipient) are in the same IS. *Line 2* shows that user *PC305* has sent a message which contains all the required parameters to the *IMNreq* ambient to deliver a message to user *Phone306*. IS1 now will only sends requests to the ISC to validates both users. So, from *line 5* to *line 10* the ISC validates the sender, and from *line 12* to *line 17* it validates the recipient. *Line 19* shows that the recipient has received the message.

However, Figure 6.16 shows the execution and animation of two users in different ISs exchanging messages. Now, because the recipient is not in the same IS as the sender, the IS sends the whole request to the ISC to deal with. In *line 4*, IS1 sends the request to ISC with a flag set to 5 indicating that the ISC should validate both users and deliver the message to the recipient. From *line 5* to *line 14* the ISC validates both users. In *lines 15 - 19*, the ISC fetches the location of the recipient, which is IS2 in this scenario. In *line 20* the ISC sends the message to the corresponding IS, and *line 22* shows the recipient *Phone306* has received the message.

6.4 Summary

In this chapter, we presented the execution environment of `CCA`. We first presented the architecture of the execution environment, followed by the `ccaPL` notation which is used to write `CCA` executable programs. Then we demonstrated some examples on how to execute a `CCA` program. We then introduced the `ccaPL` animator which is a graphical aid to the `CCA` interpreter.

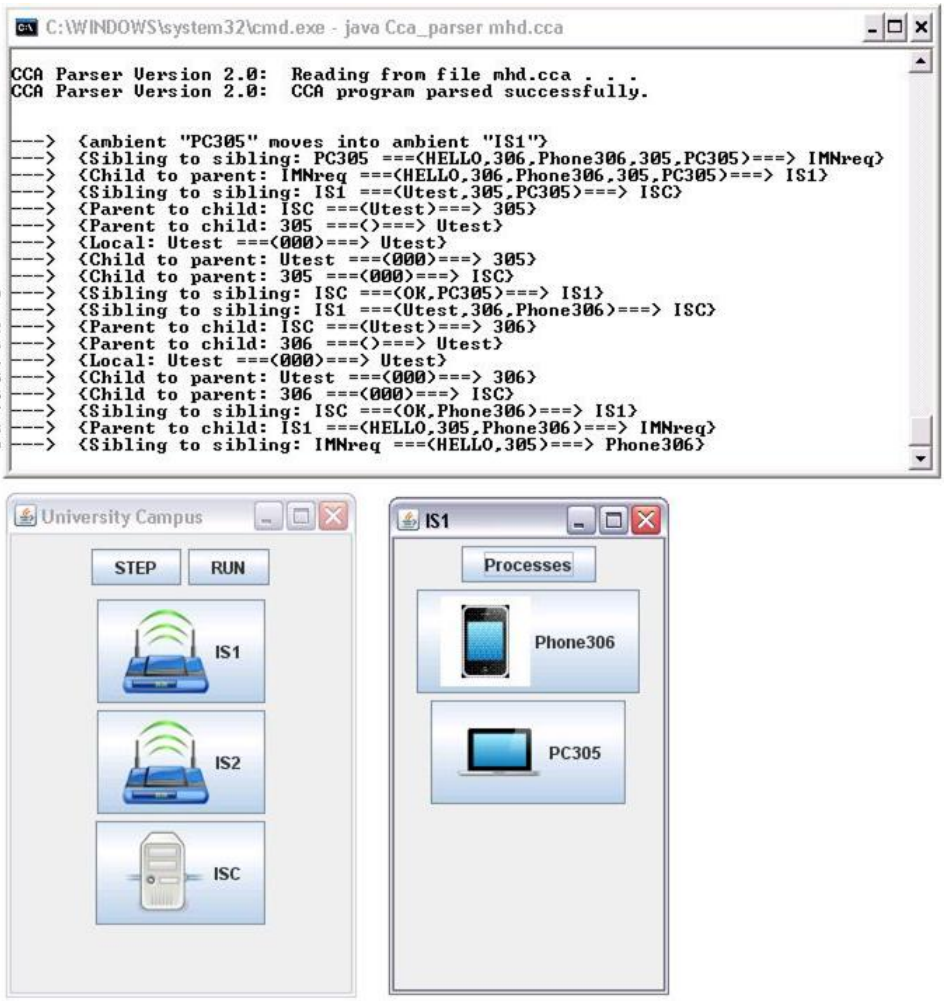


Figure 6.15: Execution and animation of the IMN service: sender and recipient in the same IS

Finally, we validated some properties regarding the infostation-based mLearning system, moreover, we showed the execution and animation of each validated property.

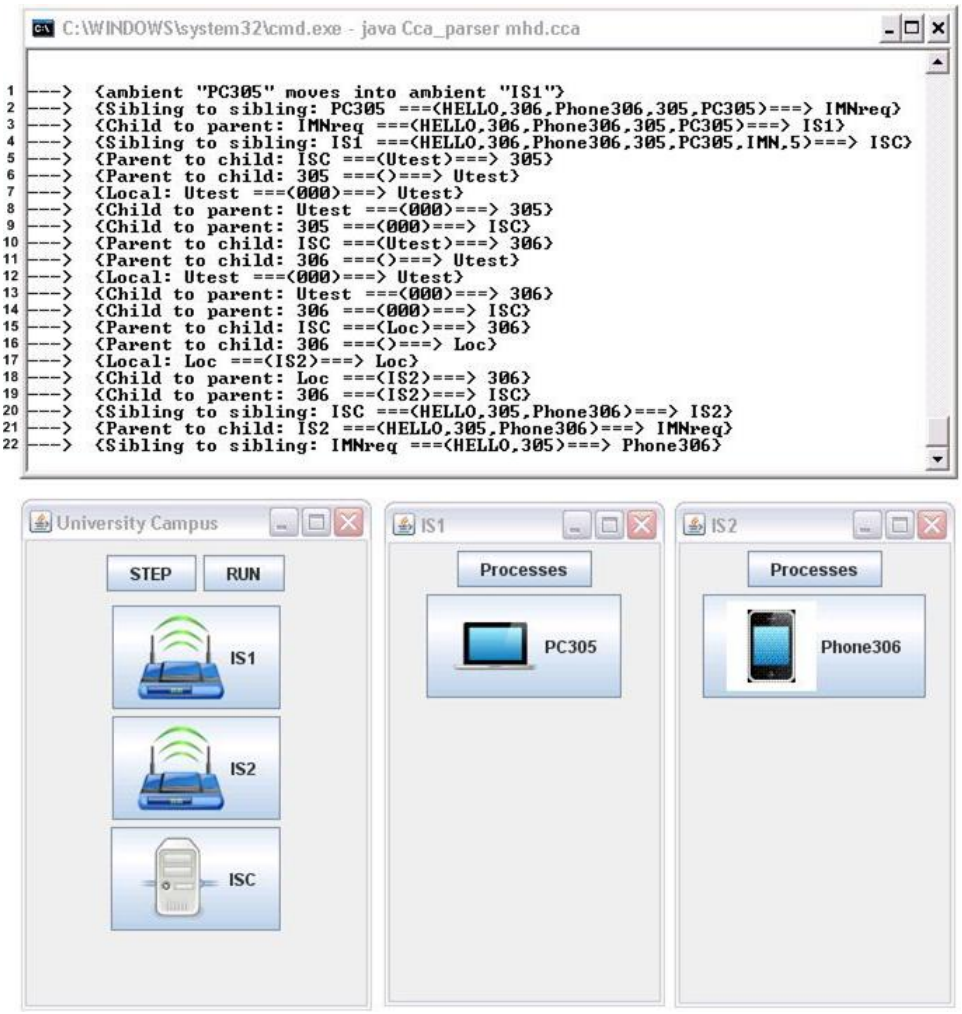


Figure 6.16: Execution and animation of the IMN service: sender and recipient in different ISs

Chapter 7

Conclusion and Future Work

Objectives:

- Summarise the work in this thesis
- Give the statement of evaluation
- List the main contributions of this work
- Give the future work that follows from this thesis

7.1 Research Summary

The work in this thesis can be summed up as follows: we provided a background study on context, context-awareness and context-aware systems. We then presented the phases of the development cycle of a context-aware system and have identified some of the problems associated with each one. Also, we presented an overview on policies, their definitions, terminology, types and uses. At the last of this chapter, we presented the ECA model which is used to represent usually the management policies which often contains an *if* part and a *then* part (Chapter 2).

Afterwards, we presented a computational model which we used to describe the work in this thesis. Firstly, the computational model is ambient-based (i.e. based on the notion of ambient). Then, we presented the mathematical notation of `CCA`, in which we presented the syntax of `CCA` processes and capabilities. We also gave some examples on how to model in `CCA`. Finally, we talked about how policies in general and the ECA model in specific can be specified using the notion of `CCA`, in addition to giving the specification of a context-aware policy in `CCA` (Chapter 3).

In the following chapter we proposed a novel conceptual policy-based architecture for the development of context-aware systems. In this approach, we integrated the policy-based management approach with a context-aware system's architecture, and we explained each component of the proposed approach. Such an approach allows a context-aware system to be modelled with regards to its policies. At the end of this chapter, we gave an abstract specification of the policy enforcer of our approach (Chapter 4).

At this stage we started the procedure of our approach's evaluation. We first

presented briefly a real-world case study of an infostation-based mLearning system (Full details are given in Appendix B). The second stage of the evaluation elaborates using CCA to formally specify the infostation-based mLearning system. We first constructed a model for the mLearning system based on the notion of ambient, then we specified the policies of the mobile services based on our modelling of the mLearning system using the mathematical notation of CCA (Chapter 5).

Finally, in the last stage of the evaluation we presented the tools and the techniques that helped us validate the mLearning system's specification. At first, we presented the syntax of ccaPL which is a machine readable version of the CCA syntax, then we presented the execution environment of ccaPL. Afterwards, we proposed an animation tool for the CCA interpreter which is a graphical aid to help visualise the execution of the mLearning system. Finally, using the CCA interpreter and the animation tool, we validated some properties (in terms of 'liveness' and 'safety') regarding the mLearning services (Chapter 6).

7.2 Statement of Evaluation

There exist many formal methods that support context-awareness and/or mobility, however, most of them were not suitable to model context-aware mobile applications. For example, CONAWA [59, 58] has been proposed as a calculus for context-aware applications that is inspired by the π calculus. The syntax of CONAWA focuses on constructs which makes it possible to describe and navigate through context. Milner [73] has proposed Bigraphs as a unifying framework to model concurrent mobile systems, however, it lacks the support of context-awareness. Regarding the case study we have chosen in this thesis. [45] has proposed a UML specification of the infostation-based mLearning system. Although UML specification provides a num-

ber of advantages such as code generation and other system analyses using assistant tools, the lack of formal reasoning support is one of its limitations for the design of critical systems. As a result, we chose `CCA` to model a context-aware system, as it is a mathematical notation that supports context-awareness and mobility, and treats those primitive constructs as first-class citizens.

As part of the evaluation for our approach, we should talk about what policy-based management has added to the development process of a context-aware system. So, from a management point of view, policy-based management aims at simplifying the complexity of a system's management. That is, the integration of a policy-based management into our context-aware system's architecture has added the following features:

- *The ability to define standard policies:* defining policies allows the system to choose the rules which define its behaviour. Defining the rules of a system in the form of sets of policies helps reduce the complication of controlling the system.

- *The ability to selectively enforce policies:* with policy-based management, a system is able to choose on what entities to perform certain actions, how to and when to perform certain actions based on pre-defined events and conditions.

- *The ability to automate policy checking and enforcement:* deploying policy-based management in a system allows it to specify patterns (i.e. policy conditions) of behaviours and consequent actions, in which, whenever a policy condition is satisfied, perform the consequent action. This however helps reduce the complexity of controlling the system as well.

- *The ability to configure policies:* policies, in general, can be defined and represented using simple policy models, this helps easily and quickly the configura-

tion of some policies when they are found to have errors or out of compliance.

Moreover, in regards to the case study we have chosen, it is provably correct. We modelled the case study based on our proposed architecture and specified it using the mathematical notation of `CCA`, then used the execution environment of `ccaPL` to validate some property-based scenarios.

7.3 Contribution to Knowledge

The main contribution of the work reported in this thesis can be summarised in order as follows:

1. Developing a conceptual policy-based architecture for context-aware systems.
2. Declaring how policies can be formalised using `CCA`.
3. Modelling, formalising and analysing a real-world case study of an infostation-based mLearning system.
4. Developing an animation tool for the execution environment of `ccaPL`.
5. Proving wanted properties of the infostation-based mLearning system.

7.4 Future Work

Context-aware computing is still an active research area that has many interesting topics to be covered. In accordance to the work done in this thesis, one future work direction is to extend our approach to integrate security mechanisms, techniques and constraints within to come up with *secure policy-based architecture for context-aware systems*. Such an approach requires an extensive research on security mechanisms in general and how to secure a dynamic context-aware systems in specific. However,

another future work in regards to the work done here is to specify how policies can be best specified, modelled and managed within a system that considers context-awareness and mobility.

Regarding the infostation-based mLearning system, there still are some services (VoIP and multimedia messages) that have not been formalised yet, in addition to some aspects of eLearning in general, such as costing and billing of services to users (students, members of staff or others). However, such an important and real world system as the mLearning can be extended to be deployed around many universities, where users can use the resources of another university (such as: libraries) and move around different campuses. This brings the challenge of synchronising the specifications of many systems.

Another future work direction within the scope of `CCA` is further development of the animator of `ccaPL` execution environment to improve its layout to the extent of making it able to visualise any entity in a system, give those entities advanced animations and to enhance more features to the animation environment. Moreover, a model checker which validates a model of a system against wanted specification(s) can be developed to be added to the execution environment. A runtime verification tool can also be developed to the `CCA` execution environment, such a tool enhances the process of detecting errors by observing the behavior of the system during its normal operations. However, the notation of `CCA` is progressively improving and some notions are under development to be added to the syntax such as: time and probability.

Bibliography

[1] http://www.itu.dk/research/pls/wiki/index.php/A_Brief_Introduction_To_Bigraphs Retrieved on 25.03.2009.

[2] http://en.wikipedia.org/wiki/Policy Retrieved on 07.09.2010.

[3] Internet Engineering Task Force (IETF) http://www.ietf.com Retrieved on 10.02.2011.

[4] http://javacc.java.net/.

[5] G. D. Abowd, C. G. Atkeson, J. Hong, S. Long, R. Kooper, and M. Pinkerton. Cyberguide: A Mobile Context-Aware Tour Guide, 1996.

[6] G. D. Abowd, A. K. Dey, R. Orr, and J. Brotherton. Context-awareness in Wearable and Ubiquitous Computing. In *Journal of Virtual Reality*, pages 179–180. Springer-Verlag, 1997.

[7] H. Ailisto, P. Alahuhta, V. Haataja, V. Kyllönen, and M. Lindholm. Structuring Context Aware Applications: Five-Layer Model and Example Case (Position Paper). 2002.

[8] M. Baldauf, S. Dustdar, and F. Rosenberg. A Survey on Context-aware Systems. *Int. J. Ad Hoc Ubiquitous Computing*, 2(4):263–277, 2007.

[9] J. E. Bardram. The Java Context Awareness Framework (JCAF) - A Service Infrastructure and Programming Framework for Context-Aware Applications. In *in Pervasive Computing. 2005: Munchen*, pages 98–115, 2005.

[10] M. Beigl, A. Krohn, T. Zimmer, and C. Decker. Typical Sensors Needed in Ubiquitous and Pervasive Computing. In *in Proceedings of the First International Workshop on Networked Sensing Systems (INSS '04)*, pages 153–158, 2004.

[11] P. Bennett. *Safety Aspects of Computer Control*. Butterworth-Heinemann, 1993.

[12] G. Biegel and V. Cahill. A Framework for Developing Mobile, Context-aware Applications. In *Proceedings of the Second IEEE International Conference on Pervasive Computing and Communications (PerCom'04)*, PERCOM '04, pages 361–365. IEEE Computer Society, 2004.

[13] L. Birkedal, S. Debois, E. Elsborg, T. Hildebrandt, and H. Niss. Bigraphical Models of Context-Aware Systems. Technical Report TR-2005-74, IT University of Copenhagen, 2005.

[14] M. Bishop. *Computer Security: Art and Science*. Addison-Wesley, 2004.

[15] C. Bolchini, C. A. Curino, E. Quintarelli, F. A. Schreiber, and L. Tanca. A Data-oriented Survey of Context Models. *SIGMOD Rec.*, 36:19–26, 2007.

[16] H. Bowman, H. Cameron, P. King, and S. Thompson. Specification and Prototyping of Structured Multimedia Documents using Interval Temporal Logic. In *International Conference on Temporal Logic*. Kluwer, 1997.

[17] W. Brauer, W. Reisig, and G. Rozenberg, editors. *Proceedings of an Advanced Course on Petri Nets: Central Models and Their Properties, Advances in Petri Nets 1986-Part I*, London, UK, 1987. Springer-Verlag.

[18] M. Brown. Supporting User Mobility. In *In Proceedings of IFIP World Conference on Mobile Communications*, pages 69–77. Chapman and Hall, 1996.

[19] P. J. Brown. The stick-e Document: a Framework for Creating Context-aware Applications. pages 259–272, 1996.

[20] P. J. Brown. Triggering Information by Context. *Personal and Ubiquitous Computing*, V2:18–27, 1998.

[21] P. J. Brown, J. D. Bovey, and X. Chen. Context-aware Applications: From The Laboratory to The Marketplace. *Personal Communications, IEEE [see also IEEE Wireless Communications]*, 4(5):58–64, 1997.

[22] L. Cardelli and A. Gordon. Mobile Ambients. *Theoretical Computer Science*, 240:177–213, 2000.

[23] R. Chadha and L. Kant. *Policy-Driven Mobile Ad hoc Network Management*. Wiley-IEEE Press, 2007.

[24] H. Chen, T. Finin, and A. Joshi. Using OWL in a Pervasive Computing Broker. *Workshop on Ontologies in Agent Systems, AAMAS-2003*, 2003.

[25] T. Cioara, I. Anghel, I. Salomie, and M. Dinsoreanu. A Policy-Based Context-aware Self-Management Model. In *Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), 2009 11th International Symposium on*, pages 333 – 340, 2009.

[26] J. R. Cooperstock, K. Tanikoshi, G. Beirne, T. Narine, and W. A. S. Buxton. Evolution of a Reactive Environment. In *Proceedings of the SIGCHI conference*

on Human factors in computing systems, pages 170–177. ACM Press/Addison-Wesley Publishing Co., 1995.

[27] G. Cox, J. Serrat, J. Strassner, J. N. Souza, D. Raymer, S. Samudrala, B. Jennings, and K. Barrett. An Enhanced Policy Model to Enable Autonomic Communications. In *Proceedings of the Fifth IEEE Workshop on Engineering of Autonomic and Autonomous Systems*, pages 184–193. IEEE Computer Society, 2008.

[28] N. Damianou, A. Bandara, M. Sloman, and E. Lupu. A Survey of Policy Specification Approaches, 2002.

[29] N. Damianou, N. Dulay, E. Lupu, and M. Sloman. The Ponder Policy Specification Language. In *Lecture Notes in Computer Science*, pages 18–38. Springer-Verlag, 2001.

[30] W. Dargie. *Context-Aware Computing and Self-Managing Systems*. Chapman & Hall/CRC, 1st edition, 2009.

[31] N. Davies, K. Mitchell, K. Cheverst, and G. Blair. Developing a Context Sensitive Tourist Guide. 1998.

[32] A. Dersingh, R. Liscano, and A. Jost. Context-aware Access Control Using Semantic Policies. *Special issue on Autonomic Computing Systems and Applications ,Ubiquitous Computing and Communication Journal, UBICC*, ACSA - Special Issue:19–32, 2008.

[33] A. K. Dey. Context-aware Computing: The CyberDesk Project. In *AAAI 1998 Spring Symposium on Intelligent Environments*, pages 51–54. AAAI Press., 1998.

[34] A. K. Dey. Understanding and Using Context. *Personal and Ubiquitous Computing*, 5:4–7, 2001.

[35] A. K. Dey and G. D. Abowd. CyberDesk: The Use of Perception in Context-aware Computing. In *1st Workshop on Perceptual User Interfaces*, pages 26–27, 1997.

[36] A. K. Dey and G. D. Abowd. Towards a Better Understanding of Context and Context-awareness. In *In HUC '99: Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing*, pages 304–307. Springer-Verlag, 1999.

[37] A. K. Dey, G. D. Abowd, and D. Salber. A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-aware Applications. *Hum.-Comput. Interact.*, 16:97–166, December 2001.

[38] A. K. Dey, G. D. Abowd, and A. Wood. CyberDesk: a Framework for Providing Self-Integrating Context-aware Services. In *Proceedings of the 3rd international conference on Intelligent user interfaces*, pages 47–54. ACM, 1998.

[39] A. K. Dey, D. Salber, M. Futakawa, and G. D. Abowd. An Architecture To Support Context-Aware Applications. Technical report, 1999.

[40] A. Diller. *Z: An Introduction to Formal Methods*. Wiley, 2nd edition, 1994.

[41] S. Elrod, G. Hall, R. Costanza, M. Dixon, and J. Des Rivières. Responsive Office Environments. *Commun. ACM*, 36:84–85, 1993.

[42] P. Fahy and S. Clarke. CASS: a Middleware for Mobile Context-aware Applications. In *Workshop on Context Awareness, MobiSys*, 2004.

[43] S. Fickas, G. Kortuem, and Z. Segall. Software Organization for Dynamic and Adaptable Wearable Systems. In *Proceedings of the 1st IEEE International Symposium on Wearable Computers*, pages 56–. IEEE Computer Society, 1997.

[44] R. H. Frenkiel and T. Imielinski. Infostations: The Joy of 'many-time, many-where' Communications. Technical report, WINLAB, 1996.

[45] I. Ganchev, S. Stojanov, and D. Meere. An InfoStation-Based Multi-Agent System Supporting Intelligent Mobile Services Across a University Campus. *Journal of Computers*, 2(3), 2007.

[46] I. Ganchev, S. Stojanov, D. Meere, and M. O'Droma. InfoStation-Based mLearning System Architectures: Some Development Aspects. *Advanced Learning Technologies, IEEE International Conference on*, pages 504–505, 2008.

[47] I. Ganchev, S. Stojanov, M. O'Droma, and D. Meere. Adaptable InfoStation-based mLecture Service Provision within a University Campus. *Advanced Learning Technologies, IEEE International Conference on*, pages 165–169, 2007.

[48] I. Ganchev, S. Stoyanov, M. O'Droma, and V. Valkanova. Context-Aware mLearning Service Execution in an InfoStations Environment. *Internet and Web Applications and Services, International Conference on*, pages 632–637, 2009.

[49] I. Ganchev, S. Stoyanov, M. O'Droma, V. Valkanova, and D. Meere. Pervasive InfoStation-Based mLearning System. In *2010 Fifth International Conference on Systems and Networks Communications (ICSNC)*, pages 320–325, 2010.

[50] J. A. Goguen. Higher Order Functions Considered Unnecessary for Higher Order Programming. Technical report, SRI International, 1990.

[51] D. Grohmann and M. Miculan. Directed Bigraphs: Theory and Applications. Technical Report UDMI/12/2006/RR, University of Udine, 2006.

[52] A. M. H., A. Malatras, and G. Pavlou. A Context-aware, Policy-based Framework for the Management of MANETs. In *7th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY'06)*, 2006.

[53] T. Hofer, W. Schwinger, M. Pichler, G. Leonhartsberger, J. Altmann, and W. Retschitzegger. Context-Awareness on Mobile Devices - the Hydrogen Approach. In *Proceedings of the 36th Annual Hawaii International Conference on System Sciences (HICSS'03)*, pages 292.1–. IEEE Computer Society, 2003.

[54] H. Janicke. *The Development of Secure Multi-Agent Systems*. PhD thesis, De Montfort Univeristy, 2007.

[55] O. H. Jensen and R. Milner. Bigraphs and Mobile Processes (revised). Technical report, 2004.

[56] D. Kang, H. Lee, E. Ko, K. Kang, and J. Lee. A Wearable Context Aware System for Ubiquitous Healthcare. In *28th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pages 5192–5195, 2006.

[57] K. K. Khedo. Context-Aware Systems for Mobile and Ubiquitous Networks. In *Proceedings of the International Conference on Networking, International Conference on Systems and International Conference on Mobile Communications and Learning Technologies*, pages 123–. IEEE Computer Society, 2006.

[58] M. B. Kjẽrgaard and J. Bunde-Pedersen. CONAWA: A Formal Model for Context Awareness. Technical Report 0909-0878, Basic Research in Computer Science BRICS, 2006.

[59] M. B. Kjẽrgaard and J. Bunde-Pedersen. Towards a Formal Model of Context-awareness. In *First International Workshop on Combining Theory and Systems Building in Pervasive Computing 2006 (CTSB 2006)*, 2006.

[60] P. Korpipää and J. Mäntyjärvi. An Ontology for Mobile Device Sensor-based Context-awareness. In *Proceedings of the 4th international and interdisciplinary conference on Modeling and using context*, pages 451–458. Springer-Verlag, 2003.

[61] P. Korpipää, J. Mäntyjärvi, J. Kela, H. Keränen, and E. J. Malm. Managing Context Information in Mobile Devices. *Pervasive Computing, IEEE*, 2:42–51, 2003.

[62] G. Kortuem, Z. Segall, and M. Bauer. Context-Aware, Adaptive Wearable Computers as Remote Interfaces to 'Intelligent' Environments. In *Proceedings of the 2nd IEEE International Symposium on Wearable Computers*, pages 58–65. IEEE Computer Society, 1998.

[63] D. Kulkarni and A. Tripathi. Context-aware Role-based Access Control in Pervasive Computing Systems. In *Proceedings of the 13th ACM symposium on Access control models and technologies*, SACMAT '08, pages 113–122. ACM, 2008.

[64] P. G. Larsen. Ten Years of Historical Development Bootstrapping VDM Tools. *J-JUCS*, 7(8):692–709, 2001.

[65] S. Loke. *Context-Aware Pervasive Systems*. Auerbach Publications, 2006.

[66] E. C. Lupu and M. Sloman. Conflicts in Policy-based Distributed Systems Management. *Software Engineering, IEEE Transactions on*, 25(6):852–869, 1999.

[67] C. Mascolo, D. Ghica, M. Ryan, E. Lupu, and T. Record. UbiVal: Fundamental Approaches to Validation of Ubiquitous Computing Applications and Infrastructures.

[68] R. Milner. *A Calculus of Communicating Systems*. Springer-Verlag New York, Inc., 1982.

[69] R. Milner. The Polyadic pi-Calculus: a Tutorial. Technical report, Logic and Algebra of Specification, 1991.

[70] R. Milner. *Communicating and Mobile Systems: the Pi-Calculus*. Cambridge University Press, 1999.

[71] R. Milner. Bigraphical Reactive Systems: Basic Theory. Technical Report 523, University of Cambridge, Computer Laboratory, 2001.

[72] R. Milner. Bigraphs as a Model for Mobile Interaction. In *Graph Transformation*, volume 2505, pages 8–13. Springer Berlin / Heidelberg, 2002.

[73] R. Milner. Pure Bigraphs: Structure and Dynamics. *Information and Computation/information and Control*, 204:60–122, 2006.

[74] R. Milner. *The Space and Motion of Communicating Agents*. Cambridge University Press, 2009.

[75] H. Moungla. A Context-Aware, Policy-Based Framework for Ambient Network. In *Proceedings of the 2008 IEEE Workshop on Policies for Distributed Systems and Networks*, pages 203–206. IEEE Computer Society, 2008.

[76] Y. Oh, A. Schmidt, and W. Woo. Designing, Developing, and Evaluating Context-Aware Systems. In *Proceedings of the 2007 International Conference on Multimedia and Ubiquitous Engineering*, pages 1158–1163. IEEE Computer Society, 2007.

[77] S. Owicki and L. Lamport. Proving Liveness Properties of Concurrent Programs. *ACM Transactions on Programming Languages and Systems*, 4:455–495, 1982.

[78] G. Papamarkos, A. Poulovassilis, R. Poulovassilis, and P. T. Wood. Event-Condition-Action Rule Languages for the Semantic Web. In *In: Workshop on Semantic Web and Databases*, pages 309–327, 2003.

[79] J. Pascoe. Adding Generic Contextual Capabilities to Wearable Computers. In *Wearable Computers, 1998. Digest of Papers. Second International Symposium on*, pages 92–99, 1998.

[80] J. Pascoe, N. S. Ryan, and D. R. Morse. Human Computer Giraffe Interaction: HCI in the Field. 1998.

[81] S. Poland. *Ubiquitous Computing: Smart Devices, Environments and Interactions*. Wiley, 2009.

[82] A. Poulovassilis, G. Papamarkos, and P. Wood. Event-condition-action rule languages for the semantic web. In *Current Trends in Database Technology EDBT 2006*, volume 4254 of *Lecture Notes in Computer Science*, pages 855–864. Springer Berlin / Heidelberg, 2006.

[83] J. Rekimoto, Y. Ayatsuka, and K. Hayashi. Augment-able Reality: Situated Communication through Physical and Digital Spaces. In *Proceedings of the 2nd international symposium on wearable computers*, pages 68–75, 1998.

[84] M. Roman, C. Hess, R. Cerqueira, A. Ranganathan, R. H. Campbell, and K. Nahrstedt. A Middleware Infrastructure for Active Spaces. *IEEE Pervasive Computing*, 1:74–83, 2002.

[85] N. S. Ryan, J. Pascoe, and D. R. Morse. Enhanced Reality Fieldwork: the Context-aware Archaeological Assistant. In *Computer Applications in Archaeology 1997*. Tempus Reparatum, 1998.

[86] D. Salber, A. Dey, and G. Abowd. The Context Toolkit: Aiding The Development of Context-enabled Applications. In *CHI '99: Proceedings of the SIGCHI Conference on Human factors in Computing Systems*, pages 434–441. ACM Press, 1999.

[87] D. Salber, A. K. Dey, and G. D. Abowd. Ubiquitous Computing: Defining an HCI Research: Agenda for an Emerging Interaction Paradigm. Technical report, 1998.

[88] D. Sangiorgi and D. Walker. *PI-Calculus: A Theory of Mobile Processes*. Cambridge University Press, New York, NY, USA, 2001.

[89] J. Saperia. *SNMP at the Edge: Building Effective Service Management Systems*. McGraw Hill TELECOM, 2002.

[90] B. Schilit, N. Adams, and R. Want. Context-aware Computing Applications. In *Proceedings of the Workshop on Mobile Computing Systems and Applications*, pages 85–90. IEEE Computer Society, 1994.

[91] B. Schilit and M. Theimer. Disseminating Active Map Information to Mobile Hosts. *Network, IEEE*, 8(5):22–32, 1994.

[92] A. Schmidt, M. Beigl, and H. w. Gellersen. There is more to Context than Location. *Computers and Graphics*, 23:893–901, 1998.

[93] A. Schmidt and K. V. Laerhoven. How to Build Smart Appliances. *IEEE Personal Communications*, 8:66–71, 2001.

[94] F. Siewe. *A Compositional Framework for the Developement of Secure Access Control*. PhD thesis, De Montfort Univeristy, 2005.

[95] F. Siewe, A. Cau, and H. Zedan. CCA: A Calculus of Context-Aware Ambients. In *Advanced Information Networking and Applications*, pages 972–977, 2009.

[96] F. Siewe, H. Zedan, and A. Cau. The Calculus of Context-aware Ambients. *Journal of Computer and System Sciences*, 77(4):597 – 620, 2011.

[97] J. M. Spivey. *The Z Notation: a Reference Manual*. Prentice-Hall, Inc., 1989.

[98] T. Strang and C. Linnhoff-Popien. A Context Modeling Survey. In *In: Workshop on Advanced Context Modelling, Reasoning and Management, UbiComp 2004 - The Sixth International Conference on Ubiquitous Computing, Nottingham/England*, 2004.

[99] J. Subercaze, P. Maret, N. M. Dang, and K. Sasaki. Context-aware Applications Using Personal Sensors. In *Proceedings of the ICST 2nd international conference on Body area networks*, pages 19:1–19:5. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2007.

[100] M. N. Tahir. C-RBAC: Contextual Role-based Access Control Model. *Ubiquitous Computing and Communication Journal, UBICC*, 2:42–50, 2007.

[101] W. van der Aalst. Pi Calculus Versus Petri Nets: Let us eat "humble pie" rather than further inflate the "Pi hype". 2004. http://tmitwww.tm.tue.nl/staff/wvdaalst/pi-hype.PDF.

[102] R. Want, V. Falcao, and J. Gibbons. The Active Badge Location System. *ACM Transactions on Information Systems*, 10:91–102, 1992.

[103] M. Weiser. The Computer for the Twenty-First Century. *Scientific American*, 265(3):94–104, 1991.

[104] M. Weiser. Some Computer Science Issues in Ubiquitous Computing. *Commun. ACM*, 36:75–84, 1993.

[105] E. B. Willson. *An Introduction to Scientific Research*. Dover Publications, 1991.

[106] T. Xiaosheng, S. Qinghua, and Z. Ping. A Distributed Context-aware Model for Pervasive Service Environment. IEEE Computer Society, 2006.

[107] H. Zedan. 2010. Formal Methods Engineering Lectures for Masters Students. STRL, De Montfort University.

[108] H. Zedan, F. Siewe, and A. Cau. The Validation of Context-Aware Systems. Technical Report STRL-08-47, Software Technology Research Laboratory, 2008.

Appendix A

Examples of Predicates to CCA Context Expressions

In this appendix, we give some examples of predicates that can be used to specify common context properties such as the location of the user, with whom the user is and what resources are nearby. In these sample predicates we take the view that a process is evaluated by the immediate ambient λ say that contains it.

The following set is of *unary* predicated:

- $\texttt{has}(n) \;\widehat{=}\; \oplus\ (\bullet \mid n[\mathbf{True}] \mid \mathbf{True})$

  holds if ambient λ contains an ambient named n

- $\texttt{at}(n) \;\widehat{=}\; n[\oplus(\bullet \mid \mathbf{True})] \mid \mathbf{True}$

  holds if ambient λ is located at an ambient named n

- $\texttt{with}(n) \;\widehat{=}\; n[\mathbf{True}] \mid \oplus\,(\bullet \mid \mathbf{True})$

  holds if ambient λ is (co-located) with an ambient named n

- $\texttt{in\_with}(n) \;\widehat{=}\; \oplus\,(\bullet \mid \mathbf{True}) \mid \oplus\ (n[\mathbf{True}] \mid \mathbf{True})$

holds if ambient λ is (co-located) with the ambient n's parent

- $\texttt{out\_with}(n) \;\widehat{=}\; n[\mathbf{True}] \mid \oplus\oplus(\bullet \mid \mathbf{True})$

  holds if ambient n is (co-located) with ambient λ's parent

- $\texttt{out\_at}(n) \;\widehat{=}\; n[\oplus\oplus\ (\bullet \mid \mathbf{True})] \mid \mathbf{True}$

  holds if ambient λ is a grand-child of ambient n

- $\texttt{near}(n) \;\widehat{=}\; \texttt{has}(n) \vee \texttt{at}(n) \vee \texttt{with}(n) \vee \texttt{out\_at}(n) \vee \texttt{in\_with}(n) \vee \texttt{out\_with}(n)$

  holds if ambient n is nearby ambient λ

However, the following set of predicates is *binary*:

- $\texttt{at2}(n,m) \;\widehat{=}\; n[m[\mathbf{True}] \mid \mathbf{True}] \mid \mathbf{True}$

  holds if ambient m is located at ambient n

- $\texttt{with2}(n,m) \;\widehat{=}\; n[\mathbf{True}] \mid m[\mathbf{True}] \mid \mathbf{True}$

  holds if ambient m is with ambient n

- $\texttt{in\_with2}(n,m) \;\widehat{=}\; m[\mathbf{True}] \mid \oplus\,(n[\mathbf{True}] \mid \mathbf{True})$

  holds if ambient m will be with ambient n if m moves in

- $\texttt{out\_with2}(n,m) \;\widehat{=}\; \texttt{in\_with2}(m,n)$

  holds if ambient m will be with ambient n if m moves out

- $\texttt{out\_at2}(n,m) \;\widehat{=}\; n[\oplus(m[\mathbf{True}] \mid \mathbf{True})] \mid \mathbf{True}$

  holds if ambient m will be at ambient n if m moves out

- $\texttt{near2}(n,m) \;\widehat{=}\; \texttt{at2}(n,m) \vee \texttt{at2}(m,n) \vee \texttt{with2}(n,m) \vee \texttt{in\_with2}(n,m) \vee \texttt{out\_with2}(n,m) \vee \texttt{out\_at2}(n,m)$

  holds if ambient m is anywhere near ambient n

Following are an example of some context expressions. A satisfaction relation is denoted by the symbol $\vDash$ throughout the examples.

Example:

1. The CE $\texttt{has}(n)$ holds if the ambient which evaluates that context expression does not have a parent ambient, but has a child ambient named n.

$$sam[\odot|\ phone[\mathbf{0}]] \ \vDash\ \texttt{has}(phone)$$

where *sam* is the ambient evaluating the CE. The user/ambient Sam has no parent ambient, and contains a child ambient named *phone*, so the above CE is satisfied [1]. However, the CE $\texttt{has}(phone)$ does not satisfy the context $office[sam[\odot|\ phone[\mathbf{0}]]]$ for the evaluating ambient *sam* has a parent ambient; *office*. Instead, we can use the spatial operator $\oplus$, as follows:

$$office[sam[\odot|\ phone[\mathbf{0}]]] \ \vDash\ \oplus\texttt{has}(phone)$$

2. The CE $\texttt{at}(n)$ holds only if the ambient evaluating the context lies in a root ambient named n, for example:

$$office[sam[\odot|\ phone[\mathbf{0}]]] \ \vDash\ \texttt{at}(office)$$

3. The CE $\texttt{with}(n)$ holds if the ambient n and the ambient evaluating the context are located as roots. For example, user Sam is co-located with user Mary, as follows

$$mary[\mathbf{0}] \mid sam[\odot|\ phone[\mathbf{0}]] \ \vDash\ \texttt{with}(mary)$$

[1] See [96] for the full formal proof of this CE

As previously mentioned, if user Sam (evaluating the context) is not a root ambient in the context, then the CE will not be satisfied. Yet, we can use the spatial operator to solve this context problem, as follows:

$$\mathit{office}[\mathit{mary}[\mathbf{0}] \mid \mathit{sam}[\odot|\ \mathit{phone}[\mathbf{0}]]] \ \vDash\ \mathtt{at}(\mathit{office}) \ \wedge\ \oplus\mathtt{with}(\mathit{mary})$$

4. If user Mary is out of the office and Sam is inside, we can use the CE $\mathtt{out\_with}(n)$, which holds only if the parent of the ambient evaluating the context and ambient n are roots in the context. For example:

$$\mathit{mary}[\mathbf{0}] \mid \mathit{office}[\mathit{sam}[\odot|\ \mathit{phone}[\mathbf{0}]]] \ \vDash\ \mathtt{out\_with}(\mathit{mary})$$

The same rules apply to the remaining CEs.

Appendix B

Case Study: An Infostation-based mLearning System

B.1 Introduction

Traditional face-to-face learning methods can be replaced nowadays with alternative approaches such as eLearning. eLearning is a useful and powerful aid or a total replacement to the traditional approach of learning. The newly emerging method of mobile learning (mLearning) is being considered as a viable approach of eLearning. With mLearning the users do not have to sit in one place and learn (i.e. listen to lectures or take a test). Instead, they can do the learning process while 'on the go', i.e. adding mobility to the learning process.

Here, we consider a real-world case study of an infostation-based mLearning system [45]. The infostation paradigm was first proposed by Frenkiel *et al.* [44] as a wireless short-range communication nodes. When infostations are deployed around an area, a user can maintain uninterrupted service when moving from one infostation to another. Infostations have been used by [45] to devise an infostation-based

mLearning system which allows mobile devices, such as hand-set phones, laptops and personal digital assistants (PDAs), to access a number of services, including communicating university campus. This mLearning system provides a number of mobile services among which are: mLecture, mTutorial, mTest and communication services (private chat, intelligent message notification and phone calls).

B.2 mLearning System's Architecture

The architecture of the infostation-based mLearning system is depicted in Figures B.1 and B.2. The users within the infostation-based network can access a number of mServices through their mobile devices, via a set of infostations deployed (geographically in key points) around a university campus.

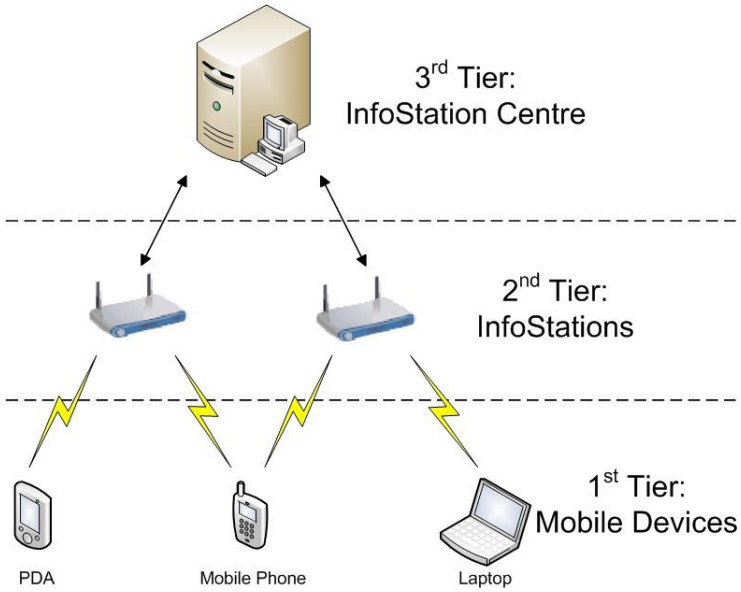


Figure B.1: The three tiers architecture of the infostation-based network [45]

The three tier network architecture consists of three essential entities: mobile devices, infostations and an infostation centre. The users (carrying mobile devices) must first register with any infostation among the system, and if successful, they

can then request a service among the available mServices. The processes of registering and requesting a service is done via the users' mobile devices. Each mService requested is delivered to the user's mobile device *eventually* in the most appropriate format to suit the user's device type (i.e. capabilities of the mobile device), regardless of their location within the system.

The *first tier* encompasses users' mobile devices (e.g. Laptop, Phone and PDA). Each mobile device runs an intelligent agent that operates as a personal assistant (PA) for the users. The *second tier* consists of Infostations (ISs in short), deployed around a university campus. These ISs enhance the operation of supplying the users with a mobile access to the mServices, through high-speed connections [45, 46]. Infostations are usually deployed irregularly (i.e. geographically) and are equipped to support different communication protocols, in order to provide successful access to a wide range of mobile devices. The *third tier* is the InfoStation Centre (ISC in short), the core of the architecture. Its main function is to control the infostations, and update and synchronise information across the system. It also manages the profiles which contain information pertaining to a specific user or a specific service.

The following gives brief information about the mobile services [45, 48]:

- **mLecture:** This service provides the users with the lectures material which they can request and access via their mobile devices.
- **mTutorial:** This service is utilised to increase the students' knowledge in a specific subject; it helps the users to take some form of self-assessment test and get the feedback for it, hence the name. This service combines both the mTest and the mLecture services.
- **mTest:** A test is crucial process to a learning process in general, so as to this mLearning system. It provides a means of evaluating the students' knowledge

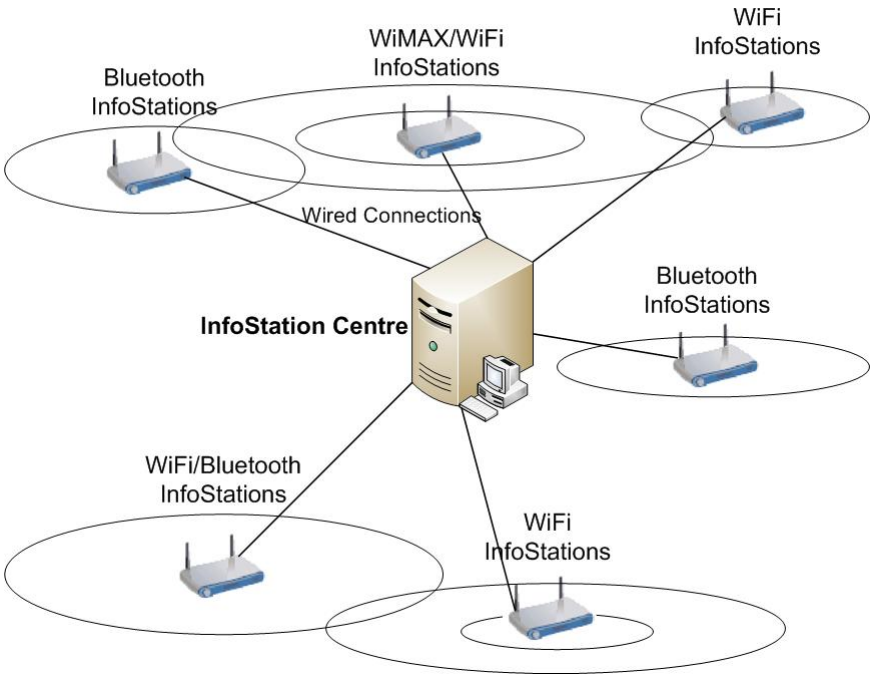


Figure B.2: The InfoStation-based network architecture

and giving a valuable feedback. The mTest allows the users to take their exams and submit the results via their mobile devices.

- **Private chat:** The idea of this service is to enable two (or more) users within the same infostation range to chat privately to each other. This is a compositional service, as it is deployed on two tiers: mobile devices and an infostation.

- **Intelligent message notification:** This service enables a user to send SMS, MMS, e-mails, etc., to another user within the range of any infostation within the system.

- **Intelligent phone calls:** This service allows users to make phone calls to each other within the range of the university campus, using *VoIP* technology.

B.2.1 User Device

A personal assistant (PA) is integrated with each user device (phone, PDA or laptop); it allows the users to register with an infostation within the system and to utilise the mServices of the mLearning system.

B.2.2 Infostation

An infostation acts as an access point through which users can register their mobile devices and request mServices. Infostations offer different wireless communication protocols such as WiFi, bluetooth, WiMAX, etc., which suits different devices' capabilities. The Authentication, Authorisation and Accounting (AAA) is a service that enables users to register their mobile devices within the system. After registering a user device, the infostation (where the user is) sends a list of the available services to the device so the user can request any of these services [45, 49].

An infostation is responsible for registering the users' devices within the system; this is done by receiving a request from a user's device, and forwarding that request to the infostation centre; where a user profile is created and the relevant information is stored. Within an infostation, there exists the catalogue of services, which is basically a list of the available services which will be forwarded to the users after registering with an infostation. The architecture of an infostation also has a cache which stores, temporarily, a copy of the requested materials (lectures and tests) for future rapid access. The infostation is responsible for responding directly to users' requests; however, if an infostation fails to deliver the requested material to a user, it forwards the request to the infostation centre which is better equipped to deal with it. For example, if a user requests a specific lecture and that lecture does not exist in

the infostation's cache, the infostation forwards the request to the infostation centre to deal with the user's request.

B.2.3 Infostation Centre

The infostation centre (ISC) works as the central management for the infostation-based mLearning system. The network topology for this system is fixed; i.e. the ISC is connected wirely to each single infostation within the system, which provides better security. The ISC has the main repository which contains profiles for each service and user. A service profile contains all the entities of that service; for example, the mLecture service profile has all the lectures material available at the mLearning system [45, 47]. A user profile has all the information related to that user: the user's name, position (i.e. student, tutor, manager, etc.), current location (i.e. infostation ID) and status (i.e. whether the user is taking a test or not).

When a user device sends an AAA request to an infostation, the infostation forwards the request to the ISC, after processing the request, if the user is authorised and authenticated, an account is created containing all the relevant information to that specific user. A positive acknowledgment is sent back to the relevant infostation, which then analyses the user profile for current user preferences and device capabilities. Afterwards, the infostation sends the acknowledgment and the mServices catalogue back to the user device (PA).

A user has a choice among the mServices of the system, when a user sends a request for a specific service, e.g. mLecture, the infostation checks whether the requested lecture exists in the infostation's cache; if not, the infostation forwards the request to the ISC to fetch the lecture [47]. In both cases, the infostation sends a request to the ISC to check whether the user is taking a test or not; this process

concerned with a critical property of the mLearning system which is: 'a user taking a test *can not* use any other service at the same time'. If, however, the infostation receives a reply from the ISC that the user is taking a test, the infostation denies the request made by the user. Eventually, if the infostation receives a reply from the ISC, saying that the user is not taking a test, the infostation then sends the requested lecture for the user.

B.3 Policies of the mServices

In this section, we present the processes inside the mServices in the form of policies. We illustrate the policies of the mLecture, mTutorial, mTest and communication services. These policies are the rules that govern the functionality of each mService, which in turn govern the functionality of the whole mLearning system.

B.3.1 mLearning Services

This section shows the learning services: mLecture, mTutorial and mTest. Each of these learning services will be presented, showing how they work and the set of policies that rules each one of them.

B.3.1.1 mLecture

The mLecture service allows users to access available lectures via their mobile devices. A user can request lecture material that is relevant to a specific subject, which is first adapted according to the user specification and device capabilities, and then delivered to the user device. The adaptation of a certain lecture to be delivered to a user is an important context-aware functionality of the mLearning system. A lecture's components can be limited to: text, pictures, audio and video. Let us suppose a lecture has the components: text, audio and video, and a user has requested

this lecture, but the user device capabilities are limited to text and audio only. The lecture has to be adapted and customised to suit the user's current device, in which case, the video component will be omitted.

Figure B.3 shows a sample interaction between the entities involved in the mLecture service provision. As mentioned earlier, when a user enters within the range of an infostation, his/her device goes through the normal AAA request procedure, and when this has been authorised and authenticated, the device receives a list of the mServices. Let us assume a user selects the mLecture service; the device sends a request to the infostation for a specific lecture; if the infostation does not have the corresponding format of the requested lecture in its cache, it forwards the request to the infostation centre, which fetches the lecture requested and sends a full copy of it back to the infostation. The infostation then starts the process of customisation of the lecture according to the user's service profile. However, the full format of the lecture may not be accessible to the user device owing to its capabilities; thus, the infostation transforms the contents of the requested lecture to the format that *best* suits the user device, and sends the customised version of the lecture to the user device. The user then can display the contents and may terminate the service at any time.

A user within an infostation range can use any type of device he wishes (phone, PDA or laptop). A user may switch devices (e.g. to a device with greater capabilities) and move to different infostation (e.g. to an infostation with better QoS) at any time, even while using a mService.

Let us suppose a user with a device, PDA say, is using the mLecture service, and decides to switch devices and then move to another infostation. When the user switches his device to another device, laptop say, the infostation re-adapts the

lecture contents to *best* suit the capabilities of the user's new device. A device change could allow the user to access the full contents of the service. After switching the device, the user wishes to move out of the infostation into another one. When this happens, the user device sends an AAA request to the new infostation and the same procedure of authorisation and authentication takes place; meanwhile, the user can still use the mLecture service un-interruptedly. As a result of users changing devices and mobility (changing infostations), there are four possible scenarios:

- No change;
- Change of infostation;
- Change of mobile device;
- Change of both.

When a user finishes using the mLecture service, the user device sends a request to the infostation to terminate that service, the infostation will forward the request to the ISC, so the ISC saves the work done by that user to keep an updated user service profile.

B.3.1.2 mTutorial

The mTutorial service allows the student to perform a self-assessment test, as it combines both the mLecture and mTest services. A user using the mTutorial service can submit the answers and receive a feedback on his progress so far. The procedure of the mTutorial service is identical to the procedure of the mLecture service. A user using the mTutorial service can switch devices and/or infostations.

B.3.1.3 mTest

A vital component to a complete learning process is the tests. The mTest service provides the students with the means of evaluating their knowledge and valuable

feedback for their progress. However, like the previous services, a user has to enter in a range of an infostation and goes through the basic AAA procedure, and then sends a request to the infostation for the mTest service.

For this service to be successful, an *on-line* and *off-line* synchronisation with the learning system is essential [45]; i.e. a user using the mTest service still has the choice of going off the infostation(s). Therefore keeping track of the user's activities is imperative to ensure the success of the whole process. Figure B.4 below depicts a sample interaction among the entities involved in the mTest service.

After being authenticated, authorised and account, a user can request the mTest service. As soon as a student accesses the mTest service, (s)he begins the test. As outlined earlier, a user taking a test is autonomous when it comes to mobility (i.e. to move from one infostation to another or even out of the system completely). In such cases, owing to the irregular geographical nature of the infostation connection, the user would probably loose contact with the system. However, the integrated PA in the user device allows the user to continue using the mTest service *uninterruptedly* while at the same time maintaining the user service profile. Thus the student may complete taking the test out of any IS's range, with the user service profile storing the student's progression.

Once the user enters the range of an infostation within the system, the same AAA procedure takes place. After the user has been authenticated, authorised and account, the infostation receives from the PA an analysed user service profile update of the user's progress for the mTest service for synchronisation (i.e. the infostation receives the completed test form for the sake of grading). The progress report received by the infostation contains additional information for the tutor of the lecture about the time spent by the student on each question. Once the grading is complete

(which is done at the infostation centre level), the student receives an assessment of the results and a feedback on his/her test performance. After this stage, the user is free to choose from any other mServices.

A golden rule governing the mTest service is that:

*A user using the mTest service **can not** use any other service simultaneously. The mTest service should be unaccompanied by any other service at all occasions.*

B.3.2 Communication Services

This subsection presents the communication services; these are private chat, messaging and personal phone calls. These services exist so that students can use them to communicate with each other and solve any problems, like any traditional learning process.

B.3.2.1 Private Chat

The private chat service uses the basic AMS service (Agent Management System) [45], which is used by agents to register/de-register to/from an infostation. As mentioned earlier, the private chat is a compositional service for it is deployed at two tiers: mobile devices and an infostation. This service allows two (or more) users to chat privately within the range of the same infostation.

After a user has been authenticated, authorised and account when arriving at an infostation, the PA on the user device register itself with the infostation. When the user requests the private chat service, the infostation send the PA a list of all the registered users within the same infostation. The user may then choose another

user among the list to chat with, remembering that the chat is private (i.e. hidden from all other users not taking part in it).

B.3.2.2 Intelligent Message Notification

The intelligent message notification (IMN in short) allows users to send SMS, MMS, e-mail, etc. to each other within the mLearning system. When a user is within the range of an infostation and after going through the AAA procedure, suppose upon a user's request, that the PA sends a request to the infostation for the IMN service, this will instantiates the service. Upon that request, the infostation will send an up-to-date virtual address book to the user's mobile device. The virtual address book has a list of all the registered users and is constantly maintained (updated by the system). The user can then select any other user (destination) among the list, and create a message in any form (s)he wishes before sending it to the nearest infostation. The infostation will analyse the recipient's profile to make an intelligent decision based on the most reliable, appropriate, quickest and cheapest way to deliver the message to the destination(s). The intelligent decision is taken based on the recipient's current location, device type and other factors found within the recipient's service profile. Figure B.5 depicts a sample interaction among the entities involved in the intelligent message notification service.

After the request has arrived at the infostation, it checks to see if the recipient is within the range of the same infostation (by analysing the recipient service profile). If the recipient is within the range, then the infostation might need to reformat the message content (to best suits the device capabilities) before sending it directly to the recipient(s). However, if the recipient is outside the range of the sender's infostation, the infostation forwards the request to the infostation centre, which in turn delivers the message to its destination. Once the message has been delivered to the

recipient(s), the infostation centre sends a positive acknowledgment to the infostation where the sender is located. The infostation then forwards the acknowledgment to the user device, and terminates the service.

B.3.2.3 Intelligent Phone Calls

The intelligent phone calls service allows registered users to make phone calls across the university campus (system network), using the VoIP technology. Like all the other services, a user has to enter an infostation range and be authenticated, authorised and account first. In this case, the user then chooses the intelligent phone calls service. The infostation then sends a virtual address book containing the names of all the registered users to the user; the user (caller) can then selects a callee's name and sends the request to the infostation. Like the intelligent message notification service, the infostation will analyse the service profile of the callee and redirects the phone call to him/her if (s)he is in the same infostation, or forwards the request to the infostation service if the callee is outside. The service profile utilisation of the callee is done based on the callee's current location and other related information in the profile (i.e. if the callee is using the mTest service, the call *can not* be forwarded to him/her), so the call has to be forwarded in the quickest, cheapest and most convenient fashion.

To establish, maintain and terminate a call, the Session Initiation Protocol (SIP) is used. There are two procedures to call initiation: *using a SIP Proxy Server* and *using a SIP Redirect Server*.

Using a SIP Proxy Server Once the user (caller) selects another user (callee) from the virtual address book, the caller's PA sends an INVITE request to the infostation. The infostation which works as the SIP proxy server in this case, receives

this request and responds with a 100 (trying) provisional message. It then retrieves the callee's current location from his/her service profile (this may require additional interaction with the ISC). There are two possible scenarios in this case, if the callee is in the range of the same infostation as the caller, the infostation forwards the INVITE message directly to him/her. However, if the callee is outside the range of the infostation, the INVITE message is forwarded to the infostation centre, which in turn will locate the callee's current location within the system and forwards the INVITE message to him/her.

After that, the callee' device will respond with a 180 (ringing) message, which the infostation will forward onto the caller. Eventually, if the callee accepts the call, a 200 (OK) message is forwarded to the infostation which will forward the message to the caller. Subsequently, the caller sends an acknowledgment (ACK) back to the callee, and a session is established between the two parties. To end the session (call), one of the users has to hang up, which results in sending a (BYE) message. When the other user receives this message, (s)he will respond with a 200 (OK) message. Upon receiving this message, the session is terminated. Figure B.6 shows the service provision of the proxy mode of the intelligent phone call.

Using a SIP Redirect Server Like the proxy server mode, a caller can selects a callee from the virtual address book, which initiates an INVITE request to the infostation (which acts as a SIP redirect server). Again, the infostation identifies the callee's current location by analysing his/her service profile. The redirect server then responds to the caller with a 302 (moved temporarily) response, which contains information about the callee's location. Afterwards, the caller responds with an ACK message and sends a fresh INVITE message to the device specifies in the redirection information. The redirection information might be the callee's device (if the callee

is in the same infostation) or through another server / infostation. Eventually if the callee responds after receiving the INVITE message, his/her PA responds with an ACK message and a session is established between the caller and callee. The termination process is exactly the same as the proxy's mode: a BYE message is sent when a user hangs up and the other responds with a 200 (OK) message to end the session. Figure B.7 shows the service provision of the redirection mode of the intelligent phone call service.

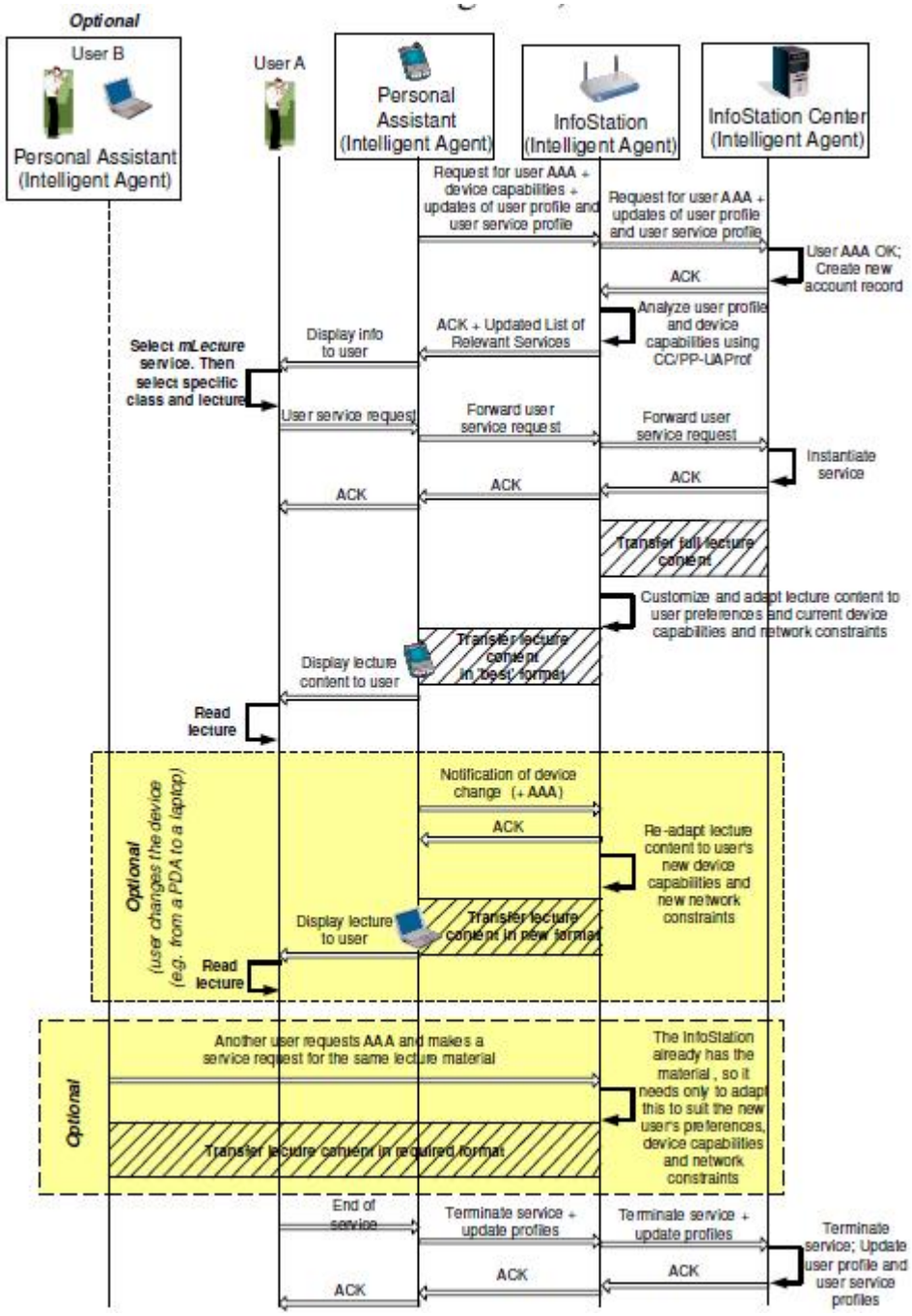


Figure B.3: mLecture service provision [45]

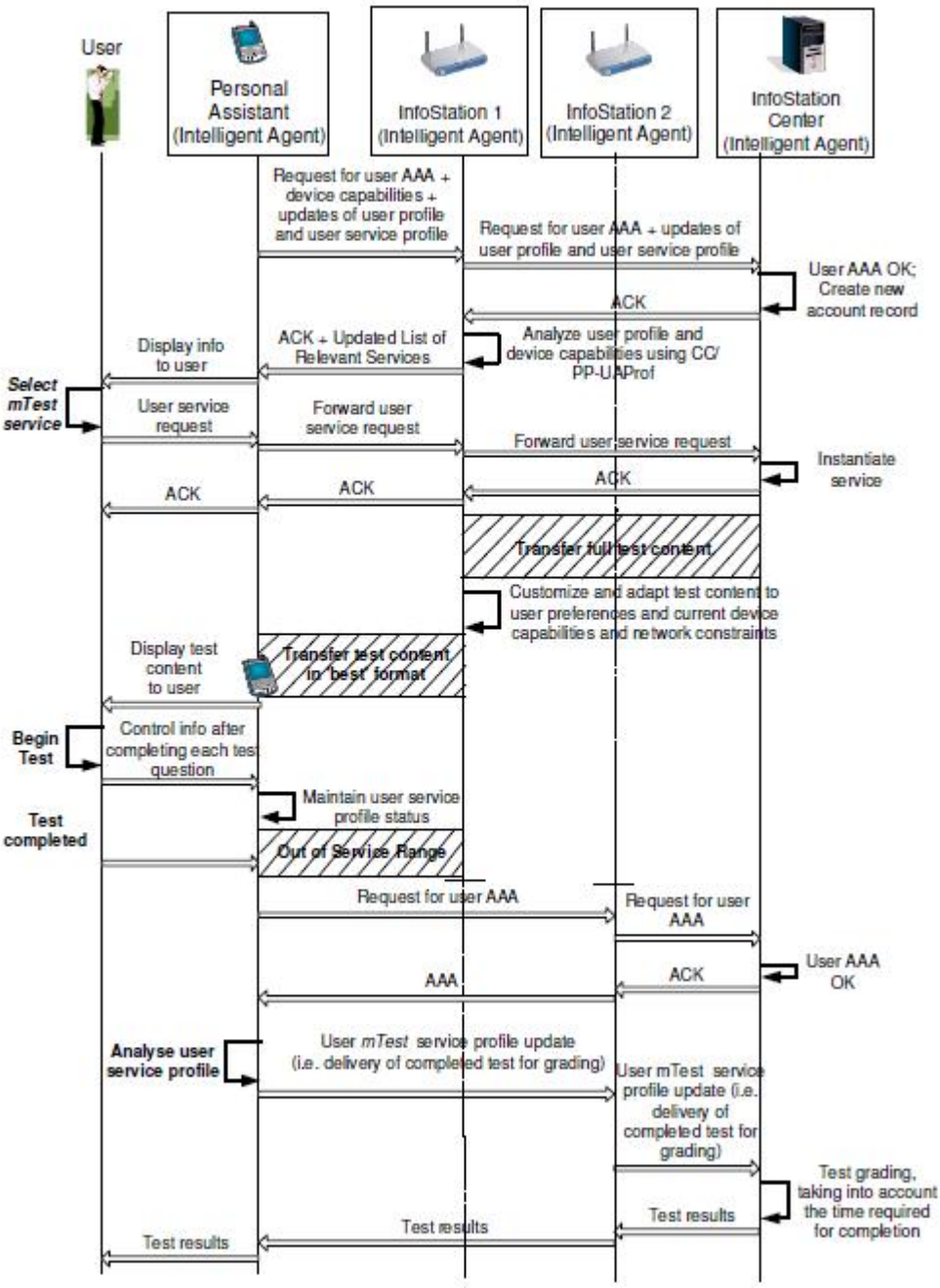


Figure B.4: mTest service provision [45]

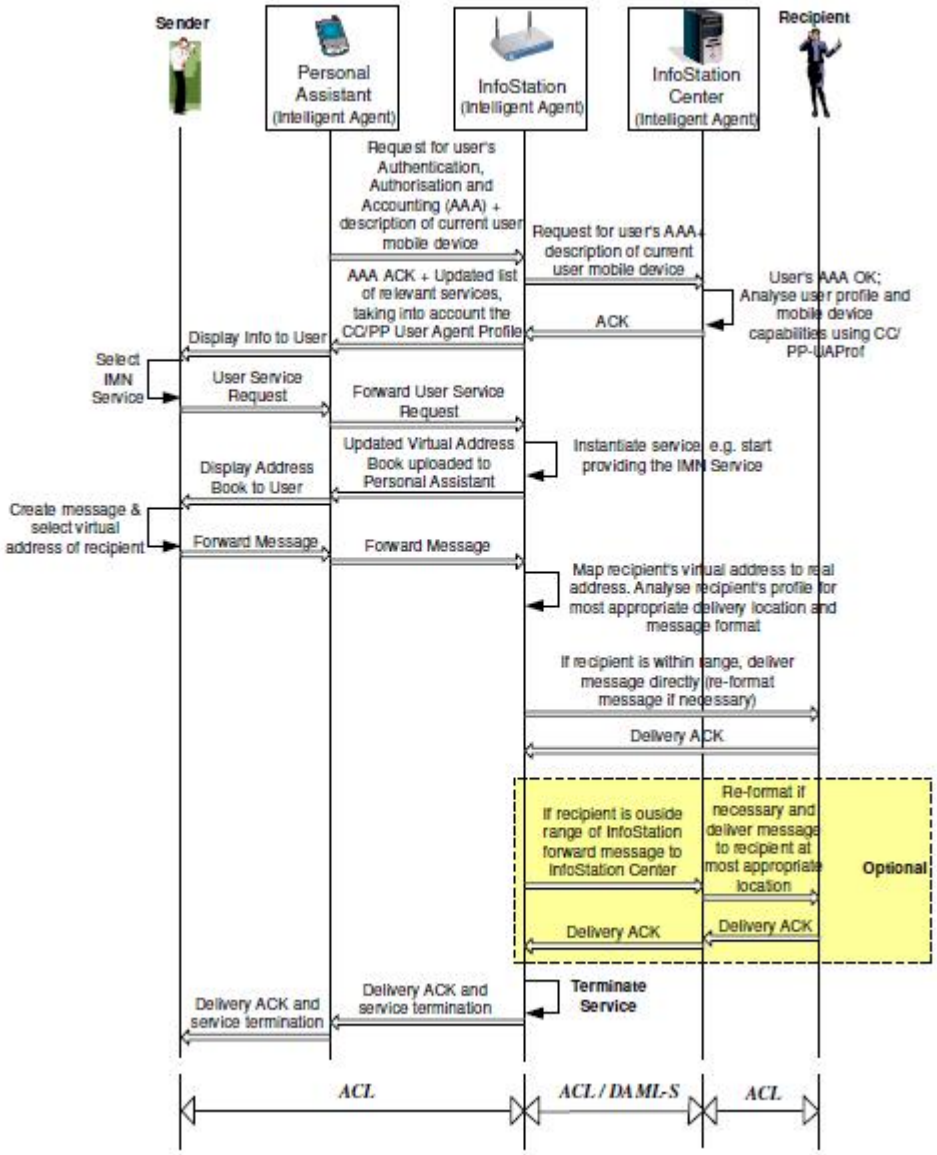


Figure B.5: Intelligent message notification service provision [45]

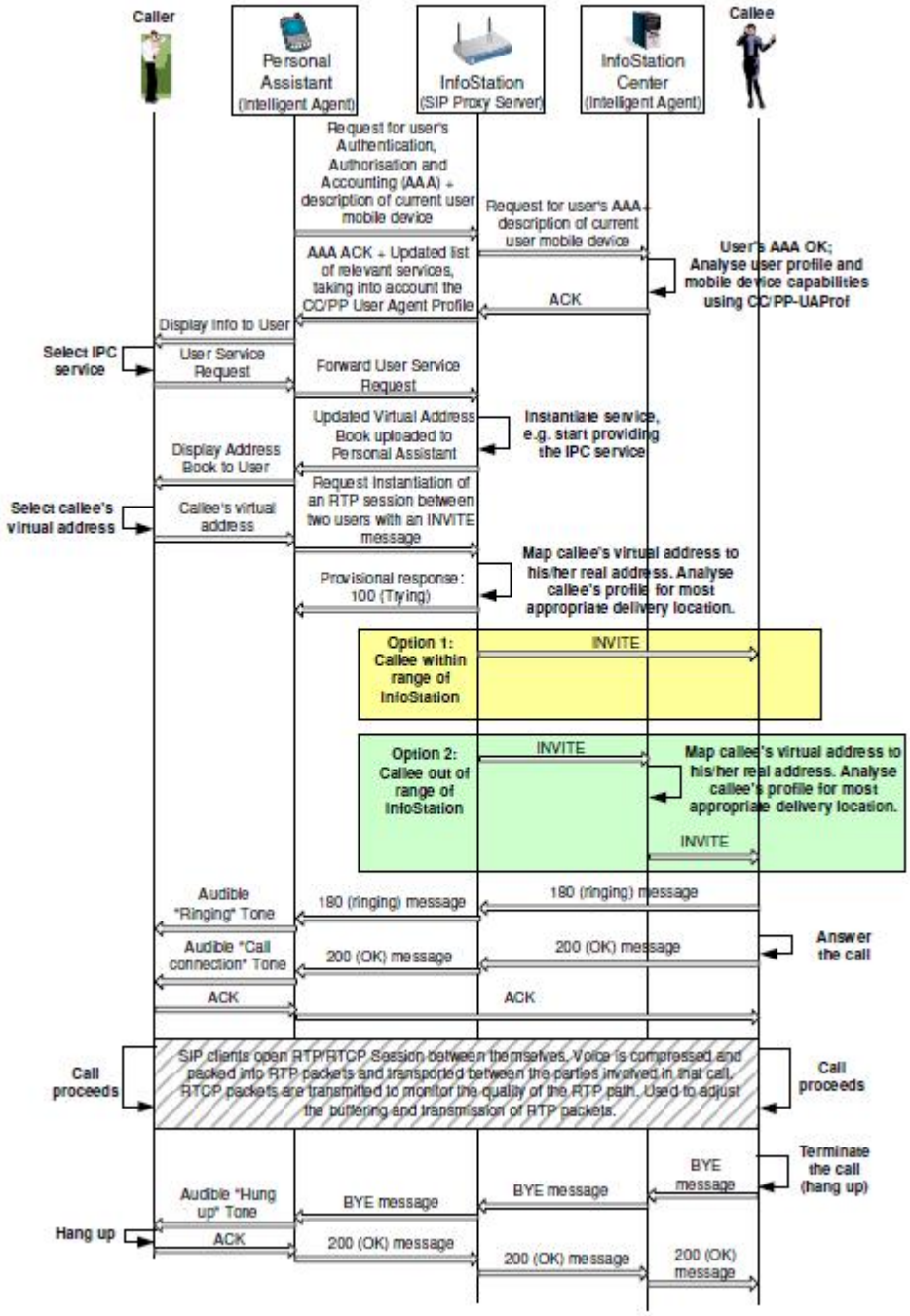


Figure B.6: Intelligent phone call service provision: proxy mode [45]

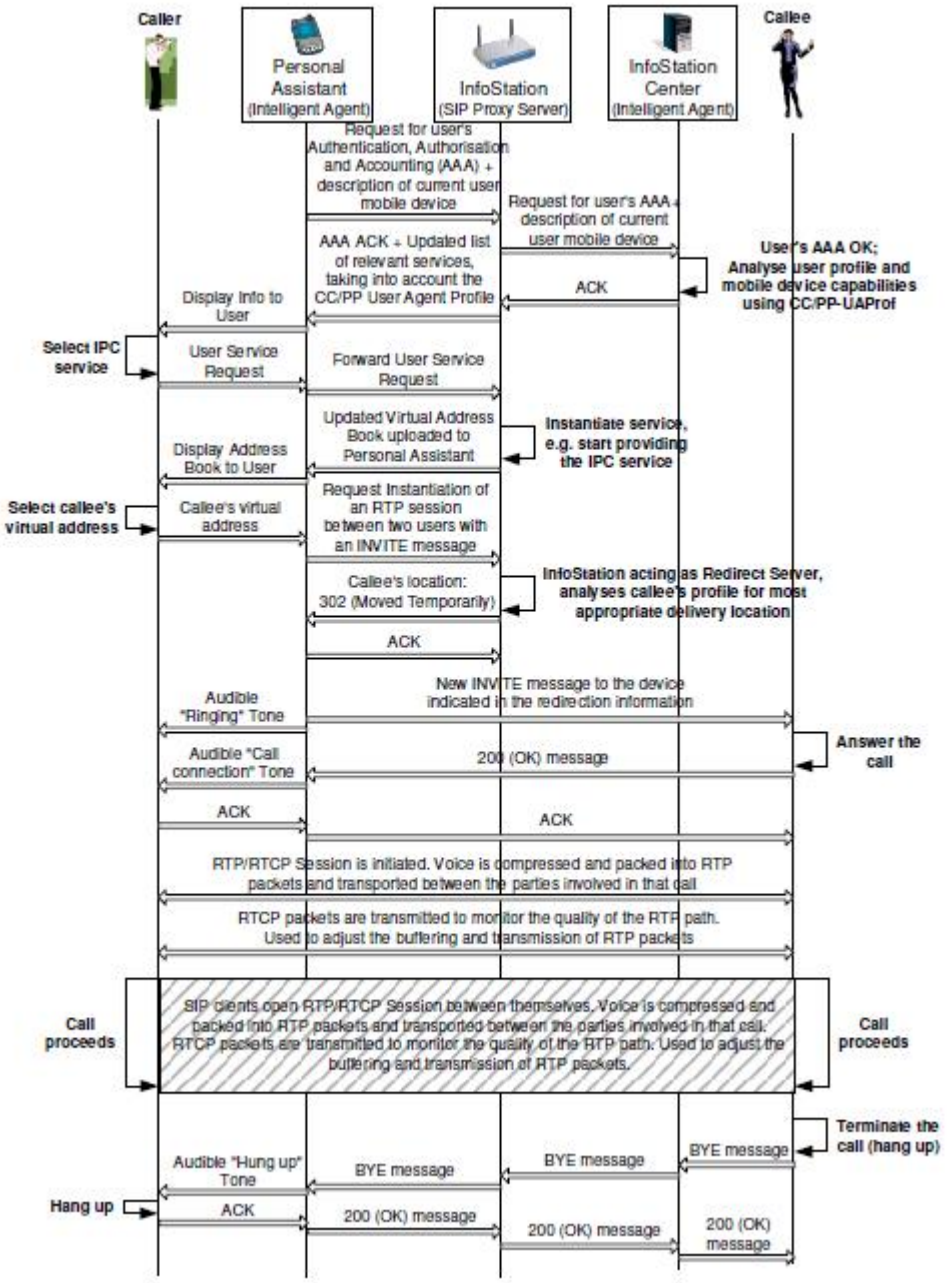


Figure B.7: Intelligent phone call service provision: redirection mode [45]

Appendix C

Formal Executable Specification of the mLearning System Using CCA

This appendix contains the full formal executable specification of the infostation-based mLearning system. The specification is composed of two infostations, namely $IS1$ and $IS2$, and one infostation centre, ISC. User behaviour can be injected in parallel with the whole specification.

```
BEGIN_DECLS
def has(n) = this | n[true] | true
def at(n) = true | n[next(this | true) | true]
//display congruence
//display code
END_DECLS

IS1
[

  AAAreq
  [
  ! :: recv(uid, dtype, aname). IS1 @ send(uid, dtype, aname). IS1 @ recv(ack, aname, slist).
  aname :: send(ack, slist).0
  ] |

  Lectreq
  [
  ! :: recv(lectid, uid, dtype, aname). IS1 @ send (lectid, uid, dtype, aname).0 |
  !IS1 @ recv (lectid, reply, aname). aname :: send(lectid, reply).0
  ] |

  Testreq
  [
  ! :: recv(testid, uid, dtype, aname). IS1 @ send (testid, uid, dtype, aname).0 |
  ! IS1 @ recv (testid, reply, aname). aname :: send(testid, reply).0
  ] |

  IMNreq
  [
  ! :: recv(msg, rid, raname, sid, saname).IS1 @ send (msg, rid, raname, sid, saname).0 |
  ! IS1 @ recv(x, y ,z). z :: send(x, y). 0
  ] |

Cache1
[
  Lect001[  Phone[  send(Content).0 |
                    ! @ recv(). recv(w). send(w).0 | @ send(w).0  |
                    ! @ recv(x). recv(y). send(x).0 | @ send().0  ] |
```

```
          PDA[    send(Content).0 |
                  ! @ recv(). recv(w). send(w).0 | @ send(w).0  |
                  ! @ recv(x). recv(y). send(x).0 | @ send().0  ] |

          PC[     send(Content).0 |
                  ! @ recv(). recv(w). send(w).0 | @ send(w).0  |
                  ! @ recv(x). recv(y). send(x).0 | @ send().0  ] |



          ! @ recv(dtype, aname).
          <somewhere has(dtype)> dtype #send(). dtype# recv(reply). @ send (reply, aname). 0 |

          <not somewhere has(dtype)> @ send(NULL, aname). @ recv(content, dtype).
          dtype # send(content).dtype #recv (). @ send(ACK). 0
          ] |

Test103[  Phone[  send(Content).0 |
                  ! @ recv(). recv(w). send(w).0 | @ send(w).0  |
                  ! @ recv(x). recv(y). send(x).0 | @ send().0  ] |

          PDA[    send(Content).0 |
                  ! @ recv(). recv(w). send(w).0 | @ send(w).0  |
                  ! @ recv(x). recv(y). send(x).0 | @ send().0  ] |

          PC[     send(Content).0 |
                  ! @ recv(). recv(w). send(w).0 | @ send(w).0  |
                  ! @ recv(x). recv(y). send(x).0 | @ send().0  ] |

          ! @ recv(dtype, aname).
          <somewhere has(dtype)> dtype #send().
          dtype# recv(reply). @ send (reply, aname). 0 |

          <not somewhere has(dtype)> @ send(NULL, aname). @ recv(content, dtype). dtype # send(content).
          dtype # recv (). @ send(ACK). 0
          ] |

! IS1 @ recv(serviceid, uid, dtype, aname, service).
<service=LECT and somewhere has (serviceid)> serviceid # send(dtype, aname).
serviceid # recv(reply, aname). IS1 @ send(serviceid, uid, dtype, reply, aname, LECT).0 |

<service=LECT and not somewhere has (serviceid)> IS1 @ send(serviceid, uid,
```

```
  dtype, NULL, aname, LECT). IS1 @ recv(serviceid, content, dtype).
  serviceid # send (content, dtype). serviceid # recv(ack). IS1 @ send(ack). 0 |

  <service=TEST and somewhere has (serviceid)> serviceid # send(dtype, aname).
  serviceid # recv(reply, aname). IS1 @ send(serviceid, uid, dtype, reply, aname, TEST).0 |

  <service=TEST and not somewhere has (serviceid)> IS1 @ send(serviceid, uid, dtype, NULL, aname, TEST).
  IS1 @ recv(serviceid, content, dtype). serviceid # send(content, dtype).
  serviceid # recv(ack). IS1 @ send(ack). 0

] |


  ! AAAreq # recv(uid, dtype, aname). ISC :: send(AAAreq, uid, dtype, aname, IS1).
  ISC :: recv(ack, aname, slist, var).
  <var=AAAreq and (somewhere has (aname))> AAAreq # send(ack, aname, slist).0
   |

  ! Lectreq # recv(lectid, uid, dtype, aname). Cache1 # send(lectid, uid, dtype, aname, LECT).0 |

  ! Testreq # recv(testid, uid, dtype, aname). Cache1 # send(testid, uid, dtype, aname, TEST).0 |

  ! Cache1 # recv(serviceid, uid, dtype, creply, aname, service).

  <service=LECT and not(creply=NULL)> ISC :: send(serviceid, uid, dtype, aname, 1, LECT).
  ISC :: recv(serviceid, reply, aname, service).
  <service=LECT and not(reply=DENIED) and somewhere has(aname)> Lectreq # send(serviceid, creply, aname).0 |
  <service=LECT and not(reply=DENIED) and not somewhere has(aname)>
  ISC :: send(serviceid, uid, dtype, aname, 00, LECT).0 |
  <service=LECT and (reply=DENIED) and somewhere has(aname)> Lectreq # send(serviceid, reply, aname).0  |

  <service=LECT and (creply=NULL)> ISC :: send(serviceid, uid, dtype, aname, 00, LECT).
  ISC :: recv(serviceid, reply, aname, service).
  <service=LECT and (not(reply=DENIED)) and somewhere has(aname)> Cache1 # send(serviceid, reply, dtype).
  Lectreq # send(serviceid, reply, aname). Cache1 # recv(ack).0 |
  <(service=LECT and not(reply=DENIED)) and not somewhere has (aname)>
  ISC :: send(serviceid, uid, dtype, aname, 00, LECT). Cache1 # recv(ack).0 |
  <service=LECT and (reply=DENIED) and somewhere has(aname)> Lectreq # send(serviceid, reply, aname).0  |

  <service=TEST and not(creply=NULL)> ISC :: send(serviceid, uid, dtype, aname, 1, TEST).
  ISC :: recv(serviceid, reply, aname, service).
```

```
  <(service=TEST) and (reply=OK) and somewhere has(aname)> Testreq # send(serviceid, creply, aname).0 |
  <service=TEST and (reply=OK) and not somewhere has(aname)>
  ISC :: send(serviceid, uid, dtype, aname, 00, TEST) |
  <service=TEST and (reply=DENIED) and somewhere has(aname)> Testreq # send(serviceid, reply, aname).0  |
  <service=TEST and (creply=NULL)> ISC :: send(serviceid, uid, dtype, aname, 00, TEST).
  ISC :: recv(serviceid, aname, reply, service).
  <service=TEST and (not(reply=DENIED)) and somewhere has(aname)> Cache1 # send(serviceid, reply, dtype).
  Testreq # send(serviceid, reply, aname). Cache1 # recv(ack).0 |
  <service=TEST and (not(reply=DENIED)) and not somewhere has (aname)> Cache1 # send(serviceid, reply, dtype).
  ISC :: send(serviceid, uid, dtype, aname, 00, TEST). Cache1 # recv(ack).0 |
  <service=TEST and (reply=DENIED) and somewhere has(aname)> Testreq # send(serviceid, reply, aname).0
   |

  ! IMNreq # recv(msg, rid, raname, sid, saname).

  <not somewhere has (raname)> ISC :: send(msg, rid, raname, sid, saname, IMN, 5).0 |
  <somewhere has (raname)> ISC :: send (Utest, sid, saname). ISC :: recv(reply, saname).
  <reply=DENIED> IMNreq # send(reply, rid, saname).0 |
  <not(reply=DENIED)> ISC :: send(Utest, rid, raname). ISC :: recv(reply, raname).
  <reply=DENIED> IMNreq # send(reply, rid, saname).0 |
  <not(reply=DENIED)> IMNreq # send(msg, sid, raname).0
   |

  ! ISC :: recv(a, b, c). IMNreq # send(a, b, c).0

] |



IS2
[

  AAAreq
  [
  ! :: recv(uid, dtype, aname). IS2 @ send(uid, dtype, aname). IS2 @ recv(ack, aname, slist).
  aname :: send(ack, slist).0
  ] |

  Lectreq
  [
  ! :: recv(lectid, uid, dtype, aname). IS2 @ send (lectid, uid, dtype, aname).0 |
  !IS2 @ recv (lectid, reply, aname). aname :: send(lectid, reply).0
```

```
] |

Testreq
[
! :: recv(testid, uid, dtype, aname). IS2 @ send (testid, uid, dtype, aname).0 |
!IS2 @ recv (testid, reply, aname). aname :: send(testid, reply).0
] |

IMNreq
[
! :: recv(msg, rid, raname, sid, saname).IS2 @ send (msg, rid, raname, sid, saname).0 |
! IS2 @ recv(x, y, z). z :: send(x, y). 0
] |

Cache2
[
  Lect002[  Phone[  send(Content).0 |
                    ! @ recv(). recv(w). send(w).0 | @ send(w).0  |
                    ! @ recv(x). recv(y). send(x).0 | @ send().0  ] |

            PDA[    send(Content).0 |
                    ! @ recv(). recv(w). send(w).0 | @ send(w).0  |
                    ! @ recv(x). recv(y). send(x).0 | @ send().0  ] |

            PC[     send(Content).0 |
                    ! @ recv(). recv(w). send(w).0 | @ send(w).0  |
                    ! @ recv(x). recv(y). send(x).0 | @ send().0  ] |

            ! @ recv(dtype, aname).
            <somewhere has(dtype)> dtype #send(). dtype# recv(reply). @ send (reply, aname). 0 |

<not somewhere has(dtype)> @ send(NULL, aname). @ recv(content, dtype).
dtype # send(content). dtype #recv (). @ send(ACK). 0
] |

! IS2 @ recv(serviceid, uid, dtype, aname, service).
<service=LECT and somewhere has (serviceid)> serviceid # send(dtype, aname).
serviceid # recv(reply, aname). IS2 @ send(serviceid, uid, dtype, reply, aname, LECT).0 |

<service=LECT and not somewhere has (serviceid)> IS2 @ send(serviceid, uid, dtype, NULL, aname, LECT).
IS2 @ recv(serviceid, content, dtype). serviceid # send (content, dtype).
```

```
  serviceid # recv(ack). IS2 @ send(ack). 0 |

  <service=TEST and somewhere has (serviceid)> serviceid # send(dtype, aname).
  serviceid # recv(reply, aname). IS2 @ send(serviceid, uid, dtype, reply, aname, TEST).0 |

  <service=TEST and not somewhere has (serviceid)> IS2 @ send(serviceid, uid, dtype, NULL, aname, TEST).
  IS2 @ recv(serviceid, content, dtype). serviceid # send(content, dtype).
  serviceid # recv(ack). IS2 @ send(ack). 0

] |


  ! AAAreq # recv(uid, dtype, aname). ISC :: send(AAAreq, uid, dtype, aname, IS2).
  ISC :: recv(ack, aname, slist, var).
  <var=AAAreq and (somewhere has (aname))> AAAreq # send(ack, aname, slist).0
   |

  ! Lectreq # recv(lectid, uid, dtype, aname). Cache2 # send(lectid, uid, dtype, aname, LECT).0 |

  ! Testreq # recv(testid, uid, dtype, aname). Cache2 # send(testid, uid, dtype, aname, TEST).0 |

  ! Cache2 # recv(serviceid, uid, dtype, creply, aname, service).

  <service=LECT and not(creply=NULL)> ISC :: send(serviceid, uid, dtype, aname, 1, LECT).
  ISC :: recv(serviceid, reply, aname, service).
  <service=LECT and not(reply=DENIED) and somewhere has(aname)>
  Lectreq # send(serviceid, creply, aname).0 |
  <service=LECT and not(reply=DENIED) and not somewhere has(aname)>
  ISC :: send(serviceid, uid, dtype, aname, 00, LECT).0 |
  <service=LECT and (reply=DENIED) and somewhere has(aname)> Lectreq # send(serviceid, reply, aname).0  |

  <service=LECT and (creply=NULL)> ISC :: send(serviceid, uid, dtype, aname, 00, LECT).
  ISC :: recv(serviceid, reply, aname, service).
  <service=LECT and (not(reply=DENIED)) and somewhere has(aname)> Cache2 # send(serviceid, reply, dtype).
  Lectreq # send(serviceid, reply, aname).Cache2 # recv(ack).0 |
  <(service=LECT and not(reply=DENIED)) and not somewhere has (aname)>
  Cache2 # send(serviceid, reply, dtype). ISC :: send(serviceid, uid, dtype, aname, 00, LECT).
  Cache2 # recv(ack).0 |
  <service=LECT and (reply=DENIED) and somewhere has(aname)> Lectreq # send(serviceid, reply, aname).0  |

  <service=TEST and not(creply=NULL)> ISC :: send(serviceid, uid, dtype, aname, 1, TEST).
```

```
  ISC :: recv(serviceid, reply, aname, service).
  <(service=TEST) and (reply=OK) and somewhere has(aname)> Testreq # send(serviceid, creply, aname).0 |
  <service=TEST and (reply=OK) and not somewhere has(aname)>
  ISC :: send(serviceid, uid, dtype, aname, 00, TEST) |
  <service=TEST and (reply=DENIED) and somewhere has(aname)> Testreq # send(serviceid, reply, aname).0  |
  <service=TEST and (creply=NULL)> ISC :: send(serviceid, uid, dtype, aname, 00, TEST).
  ISC :: recv(serviceid, aname, reply, service).
  <service=TEST and (not(reply=DENIED)) and somewhere has(aname)> Cache2 # send(serviceid, reply, dtype).
  Testreq # send(serviceid, reply, aname). Cache2 # recv(ack).0 |
  <service=TEST and (not(reply=DENIED)) and not somewhere has (aname)>
  Cache2 # send(serviceid, reply, dtype).
  ISC :: send(serviceid, uid, dtype, aname, 00, TEST). Cache1 # recv(ack).0 |
  <service=TEST and (reply=DENIED) and somewhere has(aname)> Testreq # send(serviceid, reply, aname).0
   |

  ! IMNreq # recv(msg, rid, raname, sid, saname).

  <not somewhere has (raname)> ISC :: send(msg, rid, raname, sid, saname, IMN, 5).0 |
  <somewhere has (raname)> ISC :: send (Utest, sid, saname). ISC :: recv(reply, saname).
  <reply=DENIED> IMNreq # send(reply, rid, saname).0 |
  <not(reply=DENIED)> ISC :: send(Utest, rid, raname). ISC :: recv(reply, raname).
  <reply=DENIED> IMNreq # send(reply, rid, saname).0 |
  <not(reply=DENIED)> IMNreq # send(msg, sid, raname).0
   |

  ! ISC :: recv(a, b ,c). IMNreq # send(a, b ,c).0

] |



ISC
[
  305[  Utest[  send(000).0 |
                ! @ recv(). recv(w). send(w).0 | @ send(w).0  |
                ! @ recv(x). recv(y). send(x).0 | @ send().0  ] |

        Loc[    send(4).0 |
                ! @ recv(). recv(w). recv(w). send(w).0 | @ send(w).0  |
                ! @ recv(x). send(x).0 | @ send().0  ] |
```

```
      ! ISC @ recv(t). t # send(). t # recv(y). ISC @ send(y).0 |
      ! ISC @ recv(t, n). t # send(n). t # recv(). ISC @ send(t, ACK).0 ] |

306[  Utest[  send(000).0 |
              ! @ recv(). recv(w). send(w).0 | @ send(w).0  |
              ! @ recv(x). recv(y). send(x).0 | @ send().0  ] |

      Loc[    send(IS2).0 |
              ! @ recv(). recv(w). send(w).0 | @ send(w).0  |
              ! @ recv(x). recv(y). send(x).0 | @ send().0  ] |



      ! ISC @ recv(t). t # send(). t # recv(y). ISC @ send(y).0 |
      ! ISC @ recv(t, n). t # send(n). t # recv(). ISC @ send(t, ACK).0 ] |

Lectures[
  Lect001[  Phone[  send(Content).0 |
                    ! @ recv(). recv(w). send(w).0 | @ send(w).0  |
                    ! @ recv(x). recv(y). send(x).0 | @ send().0  ] |

            PDA[    send(Content).0 |
                    ! @ recv(). recv(w). send(w).0 | @ send(w).0  |
                    ! @ recv(x). recv(y). send(x).0 | @ send().0  ] |

            PC[     send(Content).0 |
                    ! @ recv(). recv(w). send(w).0 | @ send(w).0  |
                    ! @ recv(x). recv(y). send(x).0 | @ send().0  ] |



  ! @ recv(dtype, aname).
  <somewhere has(dtype)> dtype #send(). dtype# recv(reply). @ send (reply, aname). 0 |

  <not somewhere has(dtype)> @ send(NULL, aname). @ recv(content, dtype). dtype # send(content).
  dtype #recv (). @ send(ACK). 0
  ] |

  Lect002[  Phone[  send(Content).0 |
                    ! @ recv(). recv(w). send(w).0 | @ send(w).0  |
                    ! @ recv(x). recv(y). send(x).0 | @ send().0  ] |

            PDA[    send(Content).0 |
```

```
                    ! @ recv(). recv(w). send(w).0 | @ send(w).0  |
                    ! @ recv(x). recv(y). send(x).0 | @ send().0  ] |

            PC[     send(Content).0 |
                    ! @ recv(). recv(w). send(w).0 | @ send(w).0  |
                    ! @ recv(x). recv(y). send(x).0 | @ send().0  ] |


  ! @ recv(dtype, aname).
  <somewhere has(dtype)> dtype #send(). dtype# recv(reply). @ send (reply, aname). 0 |

  <not somewhere has(dtype)> @ send(NULL, aname). @ recv(content, dtype). dtype # send(content).
  dtype #recv (). @ send(ACK). 0
  ] |

  ! @ recv(lectid, uid, dtype, aname).
  <somewhere has (lectid)> lectid # send(dtype, aname).lectid # recv(reply, aname).
  @ send(lectid, uid, dtype, reply, aname).0 |

  <not somewhere has (lectid)> @ send(lectid, uid, dtype, NULL, aname). @ recv(lectid, content, dtype).
  lectid # send(content, dtype). lectid # recv (ack). @ send(ack). 0
] |

Tests[
  Test103[  Phone[  send(Content).0 |
                    ! @ recv(). recv(w). send(w).0 | @ send(w).0  |
                    ! @ recv(x). recv(y). send(x).0 | @ send().0  ] |

            PDA[    send(Content).0 |
                    ! @ recv(). recv(w). send(w).0 | @ send(w).0  |
                    ! @ recv(x). recv(y). send(x).0 | @ send().0  ] |

            PC[     send(Content).0 |
                    ! @ recv(). recv(w). send(w).0 | @ send(w).0  |
                    ! @ recv(x). recv(y). send(x).0 | @ send().0  ] |


  ! @ recv(dtype, aname).
  <somewhere has(dtype)> dtype #send(). dtype# recv(reply). @ send (reply, aname). 0 |

  <not somewhere has(dtype)> @ send(NULL, aname). @ recv(content, dtype). dtype # send(content).
```

```
   dtype # recv (). @ send(ACK). 0 ] |

   Test104[  Phone[  send(Content).0 |
                     ! @ recv(). recv(w). send(w).0 | @ send(w).0  |
                     ! @ recv(x). recv(y). send(x).0 | @ send().0  ] |

             PDA[    send(Content).0 |
                     ! @ recv(). recv(w). send(w).0 | @ send(w).0  |
                     ! @ recv(x). recv(y). send(x).0 | @ send().0  ] |

             PC[     send(Content).0 |
                     ! @ recv(). recv(w). send(w).0 | @ send(w).0  |
                     ! @ recv(x). recv(y). send(x).0 | @ send().0  ] |


   ! @ recv(dtype, aname).
   <somewhere has(dtype)> dtype #send(). dtype# recv(reply). @ send (reply, aname). 0 |

   <not somewhere has(dtype)> @ send(NULL, aname). @ recv(content, dtype). dtype # send(content).
   dtype # recv (). @ send(ACK). 0 ] |


   ! @ recv(testid, uid, dtype, aname).
   <somewhere has (testid)> testid# send(dtype, aname). testid # recv(reply, aname).
   @ send(testid, uid, dtype, reply, aname).0 |

   <not somewhere has (testid)> @ send(testid, uid, dtype, NULL, aname). @ recv(testid, content, dtype).
   testid # send(content, dtype). testid # recv (ack). @ send(ack). 0
] |


! :: recv(aaareq, uid, dtype, aname, isi). uid # send(Loc, isi). uid # recv(t, ack).
isi :: send(ack, aname, SLIST, AAAreq).0 |

! :: recv(serviceid, uid, dtype, aname, flag, ind).
<ind=LECT> uid # send(Utest). uid # recv(y).
<y=1> uid # send(Loc). uid # recv(z). z :: send(serviceid, DENIED, aname, LECT).0 |
<(y=000) and (flag=1)> uid # send(Loc). uid # recv(z). z :: send(serviceid, OK, aname, LECT).0 |
<(y=000) and (flag=00)> Lectures # send(serviceid, uid, dtype, aname). Lectures # recv(lectid, uid,
dtype, reply, aname). uid # send(Loc).uid # recv(z). z :: send(lectid, reply, aname, LECT).0  |
```

```
  <ind=TEST> uid # send(Utest). uid # recv(y).
  <y=1> uid # send(Loc). uid # recv(z). z :: send(serviceid, DENIED, aname, TEST).0 |
  <(y=000) and (flag=1)> uid # send(Loc). uid # recv(z). z :: send(serviceid, OK, aname, TEST).
  uid # send(Utest, 1).uid # recv(utest, ack).0 |
  <(y=000) and (flag=00)> Tests # send(serviceid, uid, dtype, aname). Tests # recv(testid, uid,
  dtype, reply, aname). uid # send(Loc).uid # recv(z). z :: send(testid, reply, aname, TEST).
  uid # send(Utest, 1).uid # recv(utest, ack).0  |

  ! :: recv(msg, rid, raname, sid, saname, imn, c). <imn=IMN and c=5> sid # send(Utest). sid # recv(ysid).
  <ysid=1> sid # send (Loc). sid # recv(zsid). zsid :: send(DENIED, raname).0 |
  <ysid=000> rid # send(Utest). rid # recv(yrid).
  <yrid=1> sid # send(Loc). sid # recv(zsid). zsid :: send(DENIED, raname).0 |
  <yrid=000> rid # send(Loc). rid # recv(zrid). zrid :: send(msg, sid, raname).0
   |

  ! IS1 :: recv(utest, uid, aname). <utest=Utest> uid # send(Utest). uid # recv(y).
  <y=1> IS1 :: send(DENIED, aname).0 |
  <y=000> IS1 :: send(OK, aname).0  |

  ! IS2 :: recv(utest, uid, aname). <utest=Utest> uid # send(Utest). uid # recv(y).
  <y=1> IS2 :: send(DENIED, aname).0 |
  <y=000> IS2 :: send(OK, aname).0

] |
```